高职高专“十二五”计算机系列规划教材

C语言程序设计

主　编　黄毅斌　郑　麟
副主编　郑锦材　邱　彬　胡越梅

ZHEJIANG UNIVERSITY PRESS
浙江大学出版社

高清PDF原创
www.gqpdf.com

C语言程序设计

主　编　黄毅斌　郑　麟
副主编　郑锦材　邱　彬　胡越梅

ZHEJIANG UNIVERSITY PRESS
浙江大学出版社

更多高清PDF图书　www.gqpdf.com

内容提要

C 语言是以其灵活、高效、可移植性强等特点发展至今仍保持着强大的生命力，被大多数计算机专业作为第一程序设计基础课程。本书作者根据长期的教学经验，认真编排了教材结构，精心选择教学案例，强调实践与应用，重点讲解程序设计的思想和方法，力求培养学生的程序设计能力，同时也培养学生的独立思考能力，注重启发学生的发散思维，拓宽学生用计算机程序解决问题的思路。教材把握学习程序设计的规律和特点，注重实例教学，从实例中总结出一般规律，运用通俗易懂的文字，由浅入深、由易到难、循序渐进，力求把抽象的概念形象化，把复杂的算法简单化，让学生更加易学易懂。

本教材可以作为各类大专院校的计算机专业或非计算机专业的教学用书，以及各类培训用书，也可作为 C 语言程序自学用书。

高清PDF原创 www.gqpdf.com

图书在版编目（CIP）数据

C 语言程序设计 / 黄毅斌等主编 . —杭州 : 浙江大学出版社 , 2012 .1
ISBN 978-7-308-09539-6

Ⅰ . ①C… Ⅱ . ①黄… Ⅲ . ①C 语言—程序设计—高等职业教育—教材 Ⅳ . ①TP312

中国版本图书馆 CIP 数据核字（2012）第 002448 号

C 语言程序设计

主编　黄毅斌　郑　麟

责任编辑　吴昌雷
封面设计　俞亚彤
出版发行　浙江大学出版社
（杭州市天目山路 148 号　邮政编码 310007）
（网址 :http ://www .zjupress .com）
排　　版　杭州中大图文设计有限公司
印　　刷　杭州日报报业集团盛元印务有限公司
开　　本　787mm×1092mm　1/16
印　　张　14 .75
字　　数　383 千
版 印 次　2012 年 1 月第 1 版　2012 年 1 月第 1 次印刷
书　　号　ISBN 978-7-308-09539-6
定　　价　32 .00 元

版权所有　翻印必究　　印装差错　负责调换
浙江大学出版社发行部邮购电话(0571)88925591

高清PDF原创
www.gqpdf.com

前　言

目前软件开发语言工具大都使用可视化编程语言工具，面向对象的程序设计语言也逐步替代结构化设计语言，但是C语言仍被许多院校的计算机专业选为入门语言。这是因为C语言具有完备的高级语言特性、丰富灵活的控制和数据结构、简洁高效的语句表达、清晰的程序结构和良好的可移植性等特点，使其发展至今仍保持强大的生命力，在各个领域依然有着广泛的应用。以C语言作为入门基础语言，能学习到较全面的程序设计语言概念、语法、语句，在此基础上再学习其它语言，往往能事半功倍。

但是，把C语言当作入门语言对初学者来说也有不利的一面，它内容丰富、规则繁杂、语法灵活，尤其指针的部分不容易掌握。如果初学者没有掌握好的方法是难以较全面掌握语言设计基础的。C语言的实践性很强，如果都是从理论上或语法规则上进行介绍，学生还是难以真正掌握和理解的。

本教材从应用出发，强调实践与应用，重点讲解程序设计的思想和方法，力求培养学生的程序设计能力，同时也培养学生的独立思考能力，注重启发学生的发散思维，拓宽学生用计算机程序解决问题的思路。教材把握学习程序设计的规律和特点，注重实例教学，从实例中总结出一般规律，运用通俗易懂的文字，由浅入深、由易到难、循序渐进，力求把抽象的概念形象化，把复杂的算法简单化，让学生更加易学易懂。

本教材可以作为各类大专院校的计算机专业或非计算机专业的教学用书，以及各类培训用书，也可作为C语言程序自学用书。教材共分为10章，为使教材保持学科的系统性，我们仍保留了位运算、指针与数组、位段等章节，这部分知识点可根据实际情况选讲。本书由黄毅斌和郑麟主编，其中第1～3章由黄毅斌编写，第4章由胡越梅编写，第5、6、10章由郑麟编写，第7、9章由邱彬编写，第8章由郑锦材编写。

由于编者水平所限，书中难免有错漏之处，我们恳切希望得到读者们的宝贵意见，我们的邮箱地址为xszuoye@163.com，欢迎大家交流。

编　者

2012年1月

更多高清PDF图书　www.gqpdf.com

高清PDF原创
www.gqpdf.com

目　　录

第 1 章　学习前准备 …… 1

1.1　程序与程序设计语言 …… 1
1.1.1　什么是程序 …… 1
1.1.2　什么是程序设计语言 …… 1
1.2　C 语言的特点 …… 3
1.3　如何学习 C 语言 …… 4
1.4　在 Visual C++环境中运行 C 程序 …… 5
1.5　本章小结 …… 6

第 2 章　C 语言基础 …… 7

2.1　简单的 C 程序 …… 7
2.2　认识算术运算符 …… 9
2.3　认识整型与实型数据 …… 10
2.3.1　整型数据 …… 10
2.3.2　实型数据 …… 11
2.4　认识字符型数据 …… 12
2.4.1　字符常量 …… 12
2.4.2　字符变量 …… 14
2.4.3　字符串常量 …… 14
2.5　赋值运算符和赋值表达式 …… 15
2.5.1　赋值运算符 …… 15
2.5.2　变量初始化 …… 15
2.5.3　复合的赋值运算符 …… 15
2.5.4　自增、自减运算符 …… 16
2.6　格式化输出与输入函数 …… 17
2.6.1　格式化输出函数 printf() …… 17
2.6.2　格式化输入函数 scanf() …… 20
2.7　单个字符输入与输出函数 …… 21
2.8　本章小结 …… 22

高清PDF原创 www.gqpdf.com

课后练习 …… 22

第 3 章　分支程序设计 …… 24

3.1　基本分支程序 …… 24
3.1.1　分支程序例 …… 24
3.1.2　if 语句的基本语法 …… 25
3.2　关系运算符与逻辑运算符 …… 26
3.2.1　关系运算符 …… 26
3.2.2　逻辑运算符 …… 26
3.3　条件运算符 …… 29
3.4　不带 else 的 if 语句 …… 30
3.5　if 语句的嵌套使用 …… 31
3.6　switch 语句 …… 33
3.7　本章小结 …… 36
课后练习 …… 36

第 4 章　循环程序设计 …… 38

4.1　while 循环语句 …… 38
4.1.1　while 程序例 …… 38
4.1.2　while 语法 …… 40
4.2　do...while 循环语句 …… 41
4.3　for 循环语句 …… 42
4.4　循环语句的嵌套 …… 45
4.5　break 语句和 continue 语句 …… 46
4.5.1　break 语句 …… 47
4.5.2　continue 语句 …… 48
4.6　本章小结 …… 49
课后练习 …… 49

第 5 章　函数 …… 50

5.1　函数的基本概念 …… 50
5.1.1　函数的引入 …… 50
5.1.2　函数的概念 …… 52
5.1.3　函数的分类 …… 52
5.1.4　函数的定义 …… 53
5.2　函数参数的讨论 …… 55

更多高清PDF图书　www.gqpdf.com

5.2.1　形式参数和实际参数…… 55
5.2.2　函数的返回值…… 57
5.2.3　函数参数与带参宏的比较…… 60
5.3　函数的调用…… 63
5.3.1　函数调用的方式…… 63
5.3.2　对被调用函数的声明和函数原型…… 64
5.3.3　函数的嵌套调用…… 66
5.3.4　函数的递归调用…… 68
5.4　变量的作用域与存储类别…… 71
5.4.1　局部变量与全局变量…… 71
5.4.2　变量的存储类别…… 75
5.5　本章小结…… 77
课后练习…… 78

第6章　数组…… 79

6.1　一维数组…… 79
6.1.1　一维数组的定义…… 80
6.1.2　一维数组的引用…… 81
6.1.3　一维数组的初始化…… 82
6.1.4　一维数组的使用实例…… 83
6.2　二维数组…… 85
6.2.1　二维数组的定义…… 85
6.2.2　二维数组的引用…… 86
6.2.3　二维数组的初始化…… 87
6.2.4　二维数组的应用…… 88
6.3　字符数组…… 90
6.3.1　字符数组的定义…… 90
6.3.2　字符数组的初始化和引用…… 90
6.3.3　字符串和字符串结束标志…… 92
6.3.4　字符数组的输入输出…… 92
6.3.5　字符串处理函数…… 94
6.4　数组作函数参数…… 97
6.4.1　数组元素做函数参数…… 97
6.4.2　数组名做函数参数…… 98
6.5　本章小结…… 100
课后练习…… 100

高清PDF原创 www.gqpdf.com

第 7 章　指针 …… 101

7.1　指针和地址的概念 …… 101
7.2　变量的指针和指向变量的指针变量 …… 102
7.2.1　指针变量的定义 …… 103
7.2.2　指针变量的引用 …… 104
7.2.3　指针变量作为函数参数 …… 107
7.2.4　关于指针变量几个问题的进一步说明 …… 110
7.3　数组的指针和指向数组的指针变量 …… 113
7.3.1　指向数组元素的指针 …… 113
7.3.2　通过指针引用数组元素 …… 114
7.3.3　用数组名作函数参数 …… 119
7.3.4　指向多维数组的指针和指针变量 …… 127
7.4　返回指针值的函数 …… 133
7.5　指针数组和指向指针的指针 …… 136
7.5.1　指针数组的概念 …… 136
7.5.2　指向指针的指针 …… 139
7.6　本章小结 …… 141
7.6.1　指针数据类型的小结 …… 141
7.6.2　指针运算小结 …… 142
课后练习 …… 143

第 8 章　结构体 …… 145

8.1　结构体 …… 145
8.1.1　结构体的定义 …… 145
8.1.2　结构体变量的定义、初始化和引用 …… 146
8.2　结构体数组 …… 149
8.2.1　结构体数组的定义 …… 150
8.2.2　结构体数组的初始化和应用 …… 150
8.3　结构体类型数据的指针 …… 152
8.3.1　结构体变量的指针 …… 152
8.3.2　结构体数组的指针 …… 153
8.3.3　结构体类型数据的指针作函数参数 …… 154
8.4　用 typedef 定义类型 …… 156
8.5　用结构体处理链表 …… 158
8.5.1　链表概述 …… 158
8.5.2　处理链表所需的函数 …… 159
8.5.3　链表的建立 …… 160

高清PDF原创 www.gqpdf.com

更多高清PDF图书　www.gqpdf.com

8.5.4　链表的输出 …… 163
8.5.5　对链表的删除操作 …… 164
8.5.6　链表结点的插入 …… 166
8.5.7　链表的综合操作 …… 169
8.6　共用体 …… 170
8.6.1　共用体的定义 …… 170
8.6.2　共用体类型和变量的定义 …… 171
8.6.3　共用体变量的引用 …… 172
8.6.4　共用体类型数据的特点 …… 172
8.6.5　共用体应用举例 …… 172
*8.7　枚举类型 …… 174
8.8　本章小结 …… 177
课后练习 …… 178

第9章　位运算 …… 179

9.1　位运算符和位运算 …… 179
9.1.1　按位与“&”运算符 …… 179
9.1.2　按位或“|”运算符 …… 181
9.1.3　异或“^”运算符 …… 181
9.1.4　按位取反“～”运算符 …… 182
9.1.5　左移“<<”运算符 …… 183
9.1.6　右移“>>”运算符 …… 183
9.1.7　位运算赋值运算符 …… 184
9.1.8　不同长度的数据进行位运算 …… 184
9.2　位运算举例 …… 184
9.3　位段 …… 186
9.4　本章小结 …… 189
课后练习 …… 189

第10章　文件 …… 190

10.1　C语言文件概述 …… 190
10.1.1　C文件的定义 …… 190
10.1.2　C语言的文件指针 …… 191
10.2　C语言文件的基本操作 …… 192
10.2.1　文件的打开 …… 192
10.2.2　文件的关闭 …… 194
10.2.3　文件的读取与写入 …… 194

高清PDF原创 www.gaopdf.com

10.3 C 语言文件的常用读写函数 …… 194
10.3.1 字符读写函数 fgetc 和 fputc …… 195
10.3.2 字符串读写函数 fgets 和 fputs …… 197
10.3.3 数据块读写函数 fread 和 fwrite …… 199
10.3.4 格式化读写函数 fscanf 和 fprintf …… 201
10.3.5 文件的随机读写 …… 202
10.4 C 语言常用的文件检测函数 …… 204
10.5 本章小结 …… 204
课后练习 …… 205

附录 A 调试技巧 …… 206

A.1 程序调试的基本方法 …… 206
A.1.1 运行程序的方法 …… 206
A.1.2 调试程序的工具 …… 209
A.1.3 程序调试举例 …… 211
A.2 程序测试的基本方法 …… 216
A.2.1 程序测试的必要性 …… 216
A.2.2 程序测试举例 …… 216

附录 B 常用 ASCII 对照表 …… 219

附录 C C 语言常用库函数 …… 220

参考文献 …… 223

更多高清PDF图书 www.gqpdf.com

第 1 章

学习前准备

高清PDF原创 www.gqpdf.com

C 语言是计算机专业学生必修的一门基础程序设计语言,学好 C 语言能为计算机专业学生今后进行软件开发培养必要的程序逻辑能力,并为后继课程的深入学习奠定强有力的基础。为更好地进行 C 语言的学习,我们先来了解一些基本概念。

1.1 程序与程序设计语言

1.1.1 什么是程序

计算机本身并不具备独立思考的能力,它的优势在于它能够按照人给定的指令快速地运算,而且对于繁杂的运算,它不会感到枯燥和厌烦。那么人们要让计算机完成某一个计算任务,就必须设计好一系列的指令,然后输入到计算机,启动计算机执行这些指令。这个由人设计好的指令系列就称为程序。也就是说:程序是计算机指令的集合。

在计算机系统中,把计算过程描述为由许多命令按一定顺序组成程序,然后把程序和数据一起输入计算机内存,CPU 自动从内存中取出一条指令,对其进行译码、执行,然后按照顺序自动取出下一条指令继续译码执行,如此反复,这就是计算机自动运行程序的基本模式。

对于程序员来说,要让计算机完成某一项任务或完成某一项工作,可以编制计算机程序让计算机执行。进行程序设计大致可以按以下步骤进行:首先要分析好用户的需求,界定好程序的功能,即弄清楚程序要“做什么”的问题;其次要构建好解决问题的思路,设计好解决问题的算法;第三,用一种程序语言编写程序,实现该算法;第四,进行调试修改,直到程序正确完成所要求功能。

1.1.2 什么是程序设计语言

程序设计语言是用于书写计算机程序的语言,由一组标识记号和一组语法规则组成,是

人与计算机进行交流的工具。从语言的发展历程来看,程序设计语言已经历了四代:机器语言、汇编语言、高级语言和第四代语言。

1.机器语言

计算机能识别的指令是由 0 和 1 组成的一串代码,称为机器指令。不同的型号的 CPU 有不同的指令系统。如某种计算机的指令为 1011011000000000,它表示让计算机进行一次加法操作;而指令 1011010100000000 则表示进行一次减法操作。它们的前八位表示操作码,而后八位表示地址码。早期的程序员设计程序用的就是机器语言,但是使用机器语言设计程序存在下列弊端:

(1)繁杂琐碎的程序设计细节使程序员把大量的精力放在确保程序的正确性、高效性上,而难以进行创造性的劳动。

(2)复杂的程序设计细节使得程序的可靠性差,且开发周期长。

(3)由于用机器语言进行程序设计的思维和表达方式与人们的习惯大相径庭,只有必须经过较长时间职业训练的程序员才能掌握。

(4)程序由二进制代码构成,可读性差,不便于交流与合作。

(5)不同的计算机系统有不同的指令系统,所以针对某台计算机编写的程序难以在另一型号的计算机上运行,所以程序的可移植性和重用性很差。

2.汇编语言

这些弊端造成当时的计算机应用未能迅速得到推广。因此,人们创造了汇编语言,汇编语言用助记符代替操作码,用地址符号或标号代替地址码,比如,用“MOV”表示数据传递,用“ADD”代表加法运算等。这样用符号代替机器语言的二进制码,更利于人类记忆和理解。但是使用汇编语言编写的程序,机器不能直接识别,要由一种程序将汇编语言翻译成机器语言,这种起翻译作用的程序叫汇编程序,汇编程序是系统软件中语言处理系统软件。汇编语言同样十分依赖于机器硬件,移植性不好,但运行效率仍十分高,针对计算机特定硬件而编制的汇编语言程序,能准确发挥计算机硬件的功能和特长,程序精炼而质量高,所以至今仍是一种常用而强有力的软件开发工具。

3.高级语言

高级语言是面向用户的、基本上独立于计算机种类和结构的语言。其最大的优点是:形式上接近于算术语言和自然语言,概念上接近于人们通常使用的概念。高级语言的一个命令可以代替几条、几十条甚至几百条汇编语言的指令。因此,高级语言易学易用,通用性强,应用广泛,用高级语言开发的程序可读性较好,便于维护。同时,由于高级语言并不直接和硬件相关,其编制出来的程序移植性和重用性也要好得多,现代应用程序设计多数都是使用高级语言。

高级语言按其思维模式常被分为面向过程程序设计语言和面向对象程序设计语言。

所谓面向过程就是以要解决的问题为思考的出发点和核心,并使用计算机逻辑描述需要解决的问题和解决的方法。针对这两个核心目标,面向过程的程序设计语言注重高质量的数据结构和算法,研究采用什么样的数据结构来描述问题,以及采用什么样的算法来高效

更多高清PDF图书 www.gqpdf.com

地解决问题。在20世纪70年代和80年代,大多数流行的高级语言都是面向过程的程序设计语言,如Basic、Fortran、Pascal和C等。

面向对象(Object Oriented)不仅仅是一种程序设计语言的概念,应该说是一种全新的思维方式。面向对象的基本思想就是以一种更接近人类一般思维的方式去看待世界,把世界上的任何一个个体都看成是一个对象,每个对象都有自己的特点,并以自己的方式做事,不同对象之间存在着通讯和交互,以此构成世界的运转。用计算机专业的术语来说,对象的特点就是它们的属性,而能做的事就是他们的方法。常见的面向对象的程序设计语言包括C++和Java等。面向对象方法大大提高了程序的重用性,而且从相当程序上降低了程序的复杂度,使得计算机程序设计能够对付越来越复杂的应用需求。

1.2　C语言的特点

C语言是在20世纪70年代初问世的,1978年由美国电话电报公司(AT&T)贝尔实验室正式发表。C语言是一种结构化语言,它层次清晰,便于按模块化方式组织程序,易于调试和维护。C语言的表现能力和处理能力极强。它不仅具有丰富的运算符和数据类型,便于实现各类复杂的数据结构,它还可以直接访问内存的物理地址,进行位(bit)一级的操作。由于C语言实现了对硬件的编程操作,因此C语言集高级语言和低级语言的功能于一体。既可用于系统软件的开发,也适合于应用软件的开发。此外,C语言还具有效率高,可移植性强等特点。因此广泛地移植到了各种类型的计算机上,从而形成了多种版本的C语言。主要特点如下:

(1)简洁紧凑、灵活方便。C语言一共只有32个关键字,9种控制语句,程序书写形式自由,主要用小写字母表示。它把高级语言的基本结构和语句与低级语言的实用性结合起来。C语言可以像汇编语言一样对位、字节和地址进行操作,这大大增加了程序的灵活性。

(2)运算符丰富。C语言的运算符包含的范围很广泛,共有34种运算符。C语言把括号、赋值、强制类型转换等都作为运算符处理。从而使C语言的运算类型极其丰富,表达式类型多样化。灵活使用各种运算符可以实现在其他高级语言中难以实现的运算。

(3)数据结构丰富。C语言的数据类型有:整型、实型、字符型、数组类型、指针类型、结构体类型、共用体类型等。能用来实现各种复杂的数据结构的运算。并引入了指针概念,使程序效率更高。另外C语言具有强大的图形功能,支持多种显示器和驱动器。且计算功能、逻辑判断功能强大。

(4)是面向结构的语言。结构化语言的显著特点是代码及数据的分隔化,即程序的各个部分除了必要的信息交流外彼此独立。这种结构化方式可使程序层次清晰,便于使用、维护以及调试。C语言是以函数形式提供给用户的,这些函数可方便的调用,并具有多种循环、条件语句控制程序流向,从而使程序完全结构化。

(5)C语法限制不太严格,程序设计自由度大。虽然C语言也是强类型语言,但它的语法比较灵活,允许程序编写者有较大的自由度。

(6)C语言允许直接访问物理地址,可以直接对硬件进行操作。由于C语言允许直接访

问物理地址,可以直接对硬件进行操作,因此它既具有高级语言的功能,又具有低级语言的许多功能,能够像汇编语言一样对位、字节和地址进行操作,而这三者是计算机最基本的工作单元,可用来写系统软件。

(7)生成目标代码质量高,程序执行效率高。一般只比汇编程序生成的目标代码效率低10～20%。

(8)C 语言适用范围大,可移植性好。C 语言有一个突出的优点就是适合于多种操作系统,如 DOS、UNIX;也适用于多种机型。C 语言具有强大的绘图能力,可移植性好,并具备很强的数据处理能力,因此适于编写系统软件,三维,二维图形和动画,它也是数值计算的高级语言。

1.3 如何学习 C 语言

在讨论如何学习 C 语言之前,我们先来了解为什么要学习 C 语言,明确学习的目的是很重要的,只有明确了学习目的,才会有持久的学习动力。C 语言是很多程序员学习的第一个计算机语言,许多程序员经过了多年的开发经验后都感觉到 C 语言对于一个程序员来说是多么的重要,如果不懂 C 语言,你难以编写底层程序,如果不懂 C 语言,你难以写出优秀高效的程序,为什么说 C 语言如此重要呢?

第一,C 语言语法结构很简洁精练,程序效率高。C 语言本身只提供必要的语言特性,其他复杂一点的功能如文件处理、数学计算等等都以库函数方式提供,便于描述算法,大多数的程序员愿意使用 C 语言去描述算法本身,如果你想在程序设计方面有所建树,那么你应该学习 C 语言。

第二,C 语言的可移植性非常好。实际上 C 语言从早期开始就几乎达到了“一次编写,处处编译”,ANSI 在 1989 年统一了 C 语言标准以后(称之为 C89),只要特定平台上的编译器完整实现了 C89 标准,而且你的代码没有使用某些特殊的扩展,那么代码一定可以编译通过,再实现一下操作系统相关的函数库,C 语言的移植就是很简单的事情。

第三,C 语言能够深入系统底层,甚至直接和硬件打交道。许多的操作系统都是 C 语言写的。例如 Windows,Unix,Linux,Mac,Os/2 等,如果你将来希望能够编写内核程序,那么你必须懂 C 语言,才可以深入到这些操作系统当中去。

第四,很多新型的语言都是衍生自 C 语言,例如 C++,Java,C#,J#,perl,等等。掌握了 C 语言,可以说你就为掌握这些语言打好了基础,经过进一步的学习,你就可以用这些新型的语言去开发了,所以说 C 语言是程序设计的重要基础,许多招聘和考试都必考 C 语言。

明确了学习 C 语言的重要性之后,我们再来讨论一下如何学习 C 语言。很多初学者刚开始都有要学好 C 语言的愿望,但是经过一段时间后,渐渐对 C 语言失去了兴趣,再下来对 C 语言产生了厌倦甚至是恐惧的情绪。是 C 语言难学吗?与其说是 C 语言难学,倒不如说是这些人没有掌握好学习方法。什么才是好的学习方法呢?每个人的学习习惯和理解方式不同,适合自己的学习方法也就不同。在这里提出一点看法,仅供参考。

首先,不要浮躁,静下心来认真学习每一个细节。心态很重要,做一名出色的程序员要

更多高清PDF图书 www.gqpdf.com

耐得住性子，耐得住寂寞，学习时才能真正掌握每一个语句的每一个细节，当你成为一名程序员时，才能有缜密的逻辑思维，才能让你的程序滴水不漏。不少学习者心浮气躁，语句看了个大概，就自以为懂得差不多了，编写程序遇到些许困难就不愿再思考，急于找答案来看，一看似乎又懂了，下次再编写又不懂了。切记，心浮气躁是编程的大忌。

其次，仔细研读教材。教材的内容是经过精心组织安排的，知识点往往循序渐进，除有细致的讲解之外，还有详细的案例、习题，是初学者必备的工具。有知识点看不懂，请多找几本参考书，再仔细研读，“书读百遍，其意自现”。

第三，多阅读源代码。除书上的例题外，可以多找些源程序代码来阅读。阅读源程序时，对不懂的语句、函数要查阅相关资料及时弄懂，这样一来可以学习或巩固语句语法。另外，要弄清楚作者的解题思路，程序是思维的体现，不是简单的语句叠加，对初学者来说，有两件难事：一是解决问题的思路，也就是算法，二是如何用 C 语言实现算法。在阅读源代码时，我们总会有这方面的收获。

第四，多编写程序是掌握 C 语言的不二法门。一定要动手编写，不要以为看得懂程序代码就能写程序。很多失败的学习者就是在这里吃的亏，他们在学习教材时，能读懂所有内容，却懒于对习题进行编写训练，结果真正写起程序却不知如何下手。应该说编写程序不仅仅是一门知识，它还是一种技能，你不训练，就不能掌握，就如你能通过书本知道游泳的方法，但是，如果你不练习，最终你还是不会游泳。

第五，调试修改程序。程序写完后并不一定就正确，要让编译运行，让机器来检验它的正确性。当然，语法正确了还不能代表逻辑也正确了，应该多设计几个区间合理的输入数据，看看能否得到预期的输出结果，通过调试修改，不断完善程序。

第六，其他建议。多参与讨论，同一题目要求，往往有多种算法，参考他人的算法，拓宽自己的思路，尽量让程序的时间、空间效率提高。在学习的后期，设计一款小软件，让它能实际应用在某一方面。分章节的学习，我们学到的是零散的知识，如何综合运用，只有通过实际的开发才能体会，同时，也让我们学以致用，把学到的东西，转换成实实在在的好处。

总之，具体的学习方法因人而异，但是有些步骤是必须经历的，希望读者能尽快找到一种适合自己的学习 C 语言的方法。

1.4　在 Visual C++环境中运行 C 程序

C 程序通常可以在 Turbo C 或者 Win－Tc 环境下编译执行，但也可将它放在 Visual C++环境中运行。在 Visual C++6.0 中调试运行 C 程序的技巧在附录 A 中有详细说明，读者可以自行参阅，在这里我们只介绍基本的运行步骤。

步骤一：在“开始”菜单打开 Microsoft Visual C++6.0。

步骤二：选择“文件”菜单中的“新建”命令，在出现的界面中单击“文件”选项卡，如图 1.1 所示。

步骤三：选择“C++Source File”选项，在“文件”框中输入文件名，单击“确定”按钮。

步骤四：在程序编辑区中录入程序。

高清PDF原创 www.gqpdf.com



图 1.1 Microsoft Visual C++6.0“新建”窗口

步骤五:选择“编译”菜单中的“编译”命令,或是工具栏中的按钮。

步骤六:根据“编译”窗口中(如图 1.2 所示)的提示信息,修改程序,直至错误数为 0。

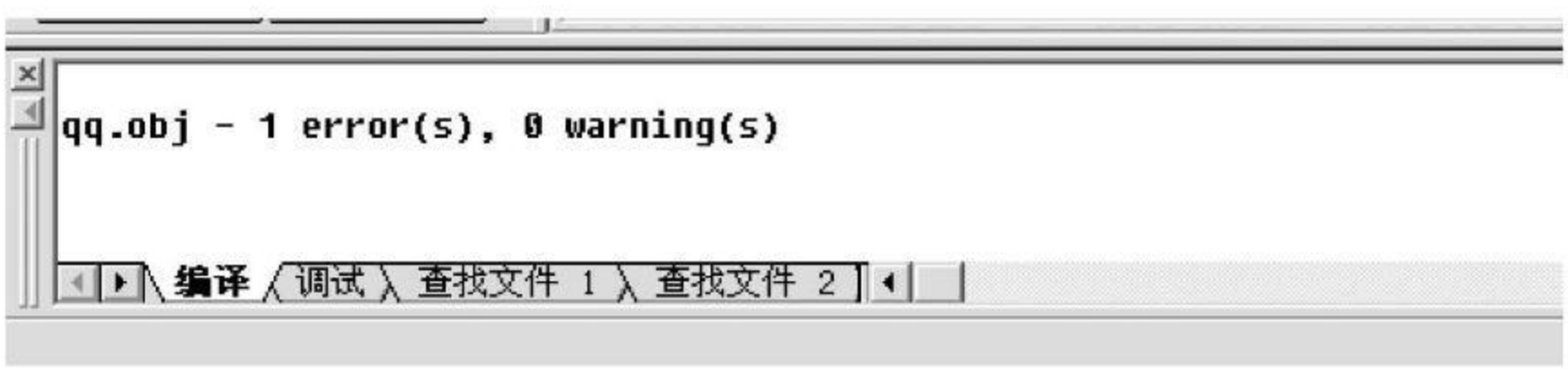


图 1.2 Microsoft Visual C++6.0“编译”窗口

步骤七:选择“编译”菜单中的“执行”命令,或是工具栏上的 按钮,执行程序。

步骤八:检查程序运行结果,如果有错,继续修改调试,直到程序结果正确。

1.5 本章小结

本章简单介绍了程序、程序设计语言的概念,介绍了 C 语言的特点和在 Microsoft Visual C++6.0 环境下运行 C 程序的方法,同时还介绍了 C 语言的学习方法,掌握一种合适的学习方法是非常重要的。

更多高清PDF图书 www.gqpdf.com

第 2 章

C 语言基础

【本章主要内容】

本章的内容比较多，主要介绍 C 程序的基本格式，算术运算符，基本数据类型（包括整型、实型、字符型），赋值运算符和赋值表达式，格式化输入与输出函数的使用。通过本章的学习，学生应能掌握简单的 C 程序写法，并对基本语法结构有所认识。

【本章要求】

(1)掌握简单 C 程序的格式。
(2)掌握算术运算符及其用法。
(3)了解整型常量与变量、实型常量与变量、字符型常量与变量的定义与使用。
(4)掌握赋值表达式的使用方法，包括复合赋值运算符、自增、自减运算符。
(5)掌握格式化输出函数 printf()函数和格式化输入函数 scanf()函数的使用方法。

2.1 简单的 C 程序

首先，请读者先运行下面这个程序，并观察它的运行结果。

【例 2.1】

【程序】

```
#include<stdio.h>
main  (    )
{int a;
 float b,z;
 printf("单价:");
```

高清PDF原创 www.gqpdf.com

```
scanf("%f",&b);
printf("数量:");
scanf("%d",&a);
z=a*b;                    /*计算金额*/
printf("金额:%.2f元\n",z);
}
```

这个程序运行后,屏幕提示“单价”,等待操作员输入一个单价后,继续提示“数量”,在操作员输入一个数量后,显示出金额——单价与数量的乘积。在 Visual C++6.0 的运行结果如图 2.1 所示。



图 2.1 程序运行结果

从上面这个程序中我们来看看写一个 C 程序的一般格式:

(1)程序必须有主函数 main()。每一个 C 源程序都必须有,而且只能有一个 main()函数。

(2)函数体在花括弧{ }之中。函数体内是命令机器执行某一功能的语句的集合,在例 2.1 的函数体中有 8 个语句。语句的多少与内容要视程序的功能不同而定。

(3)语句之间用分号“;”来分隔,例如:

```
z=a*b;
```

在 C 语言中是可以将多个语句写在同一行内的,例如:

```
printf("单价:"); scanf("%f",&b); printf("数量:");
```

养成良好的书写习惯有助于程序的调试和理解,建议大家在写程序时尽可能将不同的语句分别写在不同行,如果要写在同一行上也不应影响程序阅读。

(4)包含“库文件”。在例 2.1 中的语句:#include <stdio.h>是将头文件“stdio.h”包含进来,其原因是程序中用到的函数 printf()和 scanf()的定义在文件“stdio.h”中。C 语言将系统定义好的函数分别放在不同的文件中,这些文件一般以 .h 作为扩展名,比如程序中要使用开平方根函数 sqrt()则应在程序开头加入:#include <math.h>,其他常用函数所属的库文件可参见附录 C。注意:#include <stdio.h>之后不用加分号。

(5)必要时加入注释/*……*/。在/*和 */中的内容,机器是不编译执行的,加注释的目的是为了便于程序员理解程序,如上例中的:

```
z=a*b;                    /*计算金额*/
```

其中/*计算金额*/不是程序必需的,只是为了说明语句 z=a*b;的作用。

更多高清PDF图书 www.gqpdf.com

即时练习

下面程序的执行结果如图2.2所示，请将它补充完整，然后运行，看看是否正确。



图2.2　程序运行结果

_____①

_____②

{ printf("* \n")____③

 printf("我爱C语言！\n")____④

 printf("* \n")____⑤

_____⑥

_____⑦注释:这是一个简单程序 * /

2.2　认识算术运算符

在例2.1中表达示a * b表示a乘以b，其实在C语言的语法里，算术运算符有5个(见表2.1)。

表2.1　算术运算符

运算符	含　义	说　明	优先级
*	乘	乘法运算	高
/	除	如果除数与被除数都为整数，则结果也为整数，如5/2，结果为2；5.0/2、5/2.0或是5.0/2.0结果均为2.5。	
%	求余	求余运算只能用于整数运算，操作数中不能含有实数，如5%3结果为2，5.2%3为错误的表达式	
＋	加	单目时为正号，如＋10，表示正10	低
－	减	单目时为负号，如－10，表示负10	

若要改变算术运算的优先次序，在C语言中一律用圆括弧()。如数学表达[(5＋x)×3－4]÷2，用C语言表示成((5＋x) * 3)/2。

即时练习

将下面数学表达式转换成 C 表达式应如何描述？

$$\frac{2x+3(y+z)}{2ab+b}$$

2.3 认识整型与实型数据

在例 2.1 中有以下两个语句：

```
int a;
float b,z;
```

它们的作用是声明变量 a 是整数类型，b 和 z 是实数类型。在 C 语言中，数据量有常量与变量之分，常量是指在程序运行过程中其值不会改变的量，变量是指在程序运行过程中其值可以发生改变的量。对于变量要求要声明属于哪种类型后才能使作。常用的基本类型见表 2.2。

表 2.2 常用基本类型

类 型	声明符号
整 型	int
实 型	float 或 double
字符型	char

在这里我们先对整型与实型进行探讨。

2.3.1 整型数据

整型数据分整型常量和整型变量。

1.整型常量

C 语言的整型常量有十进制、八进制、十六进制三种表示方法：

(1)十进制，同数学表达，如 123，−89 等。

(2)八进制，以 0 开头，如 0123，表示八进制的 123，即$(123)_8$，相当于十进制的 83。

(3)十六进制，以 0X 或 0x 开头，如 0X123，表示十六进制的 123，即$(123)_{16}$，相当于十进制的 291。

更多高清PDF图书 www.gqpdf.com

2.整型变量

通常声明一个变量为某一类型,为的是确定系统要为这个变量分配多少个内存单元以及数的存储形式。整型变量通常的声明形式为:

int　＜变量名表列＞;

如:int x,y;　表示声明变量x和变量y是整型变量。根据内存空间分配的不同和表示数的范围不同,整数类型还可以细分为以下同类(见表2.3)。

表2.3　整型数据

类　型	字节数	取值范围
[signed] int	2	−32768～32767
unsigned int	2	0～65535
[signed] short [int]	2	−32768～32767
unsigned short [int]	2	0～65535
long [int]	4	−2147483648～2147483647
unsigned long [int]	4	0～4294967295

注:上表中[]内的符号可以省略;unsigned表示无符号。

定义了变量的类型后,其取值的范围也就确定了,所以我们在程序设计时要注意所定义变量的类型是否能满足需要。如有程序段:

```
int a;
a=560000;
```

其中的a=560000超出了变量a所能表示的范围,那么该程序运行时,变量a将无法正确表示该数。应该将变量a定义为长整型变量:

```
long int a;
a=560000;
```

需要说明的是,VC++中,int类型默认的宽度是4字节,这与Turbo C是不同的。

2.3.2　实型数据

实型也称为浮点型,也包括实型常量与实型变量。

1.实型常量

在C语言中,实数只采用十进制,有两种形式:十进制小数形式,指数形式。

十进制数形式:由数码0～9和小数点组成。

例如:

0.0、35.0、6.78、0.13、5.0、−123.8230

等均为合法的实数。注意,必须有小数点。

指数形式:由十进制数,加阶码标志“e”或“E”以及阶码(只能为整数,可以带符号)组成。

其一般形式为:

a E n(a 为十进制数,n 为十进制整数)

其值为 $a*10^n$。

如:

2.1E5(等于 $2.1*10^5$)

3.7E—2(等于 $3.7*10^{-2}$)

0.5E7(等于 $0.5*10^7$)

—2.8E—2(等于 $-2.8*10^{-2}$)

2.实型变量

实型变量分为:单精度(float 型)、双精度(double 型)和长双精度(long double 型)三类。

在 Turbo C 中单精度型占 4 个字节(32 位)内存空间,其数值范围为 3.4E—38～3.4E+38,只能提供七位有效数字。双精度型占 8 个字节(64 位)内存空间,其数值范围为 1.7E—308～1.7E+308,可提供 16 位有效数字。具体实数类型见表 2.4。

表 2.4 实数类型

类型说明符	比特数(字节数)	有效数字	数的范围
float	32(4)	6～7	10^{-37}～10^{38}
double	64(8)	15～16	10^{-307}～10^{308}
long double	128(16)	18～19	10^{-4931}～10^{4932}

实型变量定义与整形类似,如:

```
float x,y;          /*声明变量 x,y 为单精度类型;*/
double a,b,c;       /*声明变量 a,b,c 为双精度类型;*/
long double i,j;    /*声明变量 i,j 为长双精度类型;*/
```

高清PDF原创 www.gqpdf.com

2.4 认识字符型数据

2.4.1 字符常量

1.字符常量的表示

字符常量是用单引号括起来的一个字符。

例如:

更多高清PDF图书 www.gqpdf.com

'a'、'B'、'='、'*'、'?'

在C语言中,字符常量有以下特点:

(1)字符常量只能用单引号括起来,不能用双引号或其他括号。

(2)字符常量只能是单个字符,不能是字符串。

(3)字符在内存中存储的是其ASCII码。如果数字被定义为字符型之后参与数值运算是以其ASCII码进行运算。如:表达式3+5的结果为8,而3+'5'的结果为56,因为字符'5'的ASCII码为53。

2.转义字符

转义字符是一种特殊的字符常量。转义字符以反斜线"\"开头,后跟一个或几个字符。转义字符具有特定的含义,不同于字符原有的意义,故称"转义"字符。例如,在前面各例题printf函数的格式串中用到的"\n"就是一个转义字符,其意义是"回车换行"。转义字符主要用来表示那些用一般字符不便于表示的控制代码。

表2.5　常用的转义字符及其含义

转义字符	转义字符的意义	ASCII代码
\n	回车换行	10
\t	横向跳到下一制表位置	9
\b	退格	8
\r	回车	13
\f	走纸换页	12
\\	反斜线符"\"	92
\'	单引号符	39
\"	双引号符	34
\a	鸣铃	7
\ddd	1～3位八进制数所代表的字符	
\xhh	1～2位十六进制数所代表的字符	

广义地讲,C语言字符集中的任何一个字符均可用转义字符来表示。表中的\ddd和\xhh正是为此而提出的。ddd和hh分别为八进制和十六进制的ASCII代码。如'\101'表示字母'A','\102'表示字母'B','\134'表示反斜线,'\XOA'表示换行等。

【例2.2】 转义字符的使用。

【程序】

```
main()
{
printf("ab\tcde\rf\n");
printf("hi\\jkL\101bM\n");
}
```

【运行结果】

```
fb      cde
hi\jkLAbM
```

(为什么结果是这样,请读者自己分析)

2.4.2 字符变量

字符变量用来存储字符常量,即单个字符。字符变量的类型说明符是 char。字符变量类型定义的格式和书写规则都与整型变量相似。例如:

```
char a,b;
a='#'; b='P';
```

高清PDF原创 www.gqpdf.com

每个字符变量被分配一个字节的内存空间,因此只能存放一个字符。字符值是以 ASCII 码的形式存放在变量的内存单元之中的。所以对字符型变量的赋值可以是字符常量,也可以是该字符的 ASCII 码。如:

```
char x1;
x1='A';
```

也可以写成:

```
char x1;
x1=65;
```

因为字符'A'的 ASCII 码值为 65,它们在内存的表现形式其实是一样的。都是二进制数:

0	1	0	0	0	0	0	1

2.4.3 字符串常量

字符串常量是由一对双引号括起的字符序列。例如:"ABCD","C program","$12.5" 等都是合法的字符串常量。

字符串常量和字符常量是不同的量。它们之间主要有以下区别:

(1)字符常量由单引号括起来,字符串常量由双引号括起来。

(2)字符常量只能是单个字符,字符串常量则可以包含一个或多个字符。

(3)可以将一个字符常量赋予一个字符变量,但不能把一个字符串常量赋予一个字符变量。在 C 语言中没有相应的字符串变量。

字符常量占一个字节的内存空间。字符串常量占的内存字节数等于字符串中字节数加 1。增加的一个字节中存放字符"\0"(ASCII 码为 0)。这是字符串结束的标志(在后面的数组再做介绍)。

更多高清PDF图书 www.gqpdf.com

2.5　赋值运算符和赋值表达式

2.5.1　赋值运算符

简单赋值运算符和表达式：简单赋值运算符记为“=”。由“=”连接的式子称为赋值表达式。其一般形式为：

变量=表达式

例如：

```
x=a+b;
y=x+sqrt(b);
```

高清PDF原创 www.gqpdf.com

赋值表达式的功能是计算表达式的值再赋予左边的变量。赋值运算符具有右结合性。因此“a=b=c=5”可理解为“a=(b=(c=5))”。

在其他高级语言中，赋值构成了一个语句，称为赋值语句。而在C中，把“=”定义为运算符，从而组成赋值表达式。凡是表达式可以出现的地方均可出现赋值表达式。

例如，式子：“x=(a=5)+(b=8)”是合法的。它的意义是把5赋予a，8赋予b，再把a，b相加，最后把和赋予x，故x应等于13。

2.5.2　变量初始化

在程序给变量赋值，还可能通过在定义变量时进行，这种方法称为初始化。初始化方法的一般形式为：

类型说明符 变量1=值1，变量2=值2，…；

例如：

```
int a=3;
float b,c=5.5;
char ch1='A',ch2='P';
```

应注意，在初始化中不允许连续赋值，如“int a=b=c=5;”是不合法的。

2.5.3　复合的赋值运算符

在赋值符“=”之前加上其他二目运算符可构成复合赋值符。如+=，-=，*=，/=，%=等。

构成复合赋值表达式的一般形式为：

变量　复合运算符=表达式

它等效于

变量＝变量 运算符（表达式）

例如：

```
a+=5        等价于  a=a+5
x*=y+7      等价于  x=x*(y+7)
a%=b        等价于  a=a%b
```

2.5.4 自增、自减运算符

自增 1 运算符记为“++”，其功能是使变量的值自增 1，如 i++或++i 都是使 i 的值增 1，等价于 x=x+1。

自减 1 运算符记为“--”，其功能是使变量的值自减 1，如 i--或--i 都是使 i 的值减 1，等价于 x=x-1。

自增 1，自减 1 运算符均为单目运算，都具有右结合性。可有以下几种形式：

```
++i     /* i自增1后再参与其他运算。*/
i++     /* i参与运算后，i的值再自增1。*/
--i     /* i自减1后再参与其他运算。*/
i--     /* i参与运算后，i的值再自减1。*/
```

当自增和自减运算符出现在较复杂的表达式或语句中时，常常难于弄清，因此应仔细分析。如下例：

【例 2.3】

【程序】

```
#include<stdio.h>
main()
{
 int x1=10,x2,y1=10,y2;
 x2=++x1;
 y2=y1++;
 printf("x1=%d,x2=%d,y1=%d,y2=%d\n",x1,x2,y1,y2);
 }
```

高清PDF原创 www.gqpdf.com

【运行结果】

```
x1=11,x2=11,y1=11,y2=10
```

程序中语句“x2=++x1”；可以看成：“x1=x1+1；x2=x1；”，所以 x1 和 x2 的值均为 11。而语句“y2=y1++；”可以看成：“y2=y1；y1=y1++；”所以 y1=11 而 y2=10。也就是说++在变量的左边是先自增后操作，而++在变量的右边是先操作后自增。自减运算符--的道理与++相似，只是增 1 变为减 1，这里就不再详述了。

更多高清PDF图书 www.gqpdf.com

2.6　格式化输出与输入函数

2.6.1　格式化输出函数 printf()

在例2.1中,语句“printf("金额:%.2f元\n",z);”的作用是在屏幕上输出所计算出的金额。在C语言中,数据的输出是通过函数的调用实现的。这里先简要介绍常用的格式化输出函数printf()的基本用法。

printf函数调用的一般形式为:

printf("格式控制字符串",输出表列);

在2.1语句“printf("金额:%.2f元\n",z);”中如果z的值为391.8,则其输出的结果是:

```
金额:391.80元
```

其中的%.2f是格式控制符,%f是控制后面的变量z是以单精度类型输出,其中的“.2”是控制z的小数点后保留2位。在printf()函数中,格式控制符通常有下面几种(见表2.6)。

表2.6　printf函数常用格式控制符

格式字符	意　义
d	以十进制形式输出带符号整数(正数不输出符号)
o	以八进制形式输出无符号整数(不输出前缀0)
x,X	以十六进制形式输出无符号整数(不输出前缀0x)
u	以十进制形式输出无符号整数
f	以小数形式输出单、双精度实数
e,E	以指数形式输出单、双精度实数
g,G	以%f或%e中较短的输出宽度输出单、双精度实数
c	输出单个字符
s	输出字符串

下面我们通过两个例题来熟悉上表中的主要格式:

【例2.4】

【程序】

```
#include<stdio.h>
main()
{
 int a=97,b=98;
```

```
printf("a1=%d,b1=%d\n",a,b);
printf("a2=%o,b2=%o\n",a,b);
printf("a3=%x,b3=%x\n",a,b);
printf("a4=%c,b4=%c\n",a,b);
}
```

【运行结果】

```
a1=97,b1=98
a2=141,b2=142
a3=61,b3=62
a4=a,b4=b
```

第一行是按十进制输出,所以输出:a1=97,b1=98。
第二行是按八进制输出,将 97 和 98 转成八进制,结果输出:a2=141,b2=142。
第三行是按十六进制输出,将 97 和 98 转成十六进制,结果输出:a3=61,b3=62。
第四行是按输出以该数字为 ASCII 码的字符,结果输出:a4=a,b4=b。

【例 2.5】

【程序】

```
#include<stdio.h>
main()
{
float a=123.456;
printf("a1=%f\n",a);
printf("a2=%e\n",a);
printf("a3=%g\n",a);
}
```

【运行结果】

```
a1=123.456000
a2=1.23456e+02
a3=123.456
```

第一行,以小数形式输出;第二行,以指数形式输出,即 1.23456×10^{2};第三行:取小数或指数中最短形式输出,不输出无效的 0。

printf()函数的控制符中还有控制宽度和精度的方法,在% 和控制符间可以加入形如"m.n"的数字可以实现。见下面的例子:

【例 2.6】

【程序】

```
#include<stdio.h>
main()
{
float a=123.456;
```

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

```
printf("应收%10.2元\n",a);

}
```

【运行结果】

```
应收      123.46元
```

结果在数字123.46前有4个空格位,因为%10.2f表示输出变量a时,总宽度为10位,数字占6位,所以在数字前加4个空格。另,%10.2f还表示小数点后保留2位小数。

printf()还可以用“-”来控制其左对齐,如下例:

【例2.7】

【程序】

```
#include<stdio.h>
main()
{
float a=123.456;
printf("应收%-10.2元\n",a);

}
```

【运行结果】

```
应收123.46      元
```

运行结果与例2.6相比,4个空格出现在数字123.46后,这是因为控制符“%-10.2”要求左对齐,故变量a向左边,空格留在了右边。

即时练习

写出下面程序的运行结果并上机运行。

```
#include<stdio.h>
main()
{ int x,y;
float a,b;
x=68;y=99;
a=367.89;b=29.12;
printf("x=%d,b=%d\n",x,y);
printf("x=%o,b=%d\o",x,y);
printf("x=%c,b=%d\c",x,y);
printf("a=%-8.2f,b=%8.3f\n",a,b);
}
```

运行结果:_____。

高清PDF原创 www.gqpdf.com

2.6.2 格式化输入函数 scanf()

scanf()函数是格式输入函数,它的作用是按用户指定的格式从键盘上把数据输入指定的变量之中。

如例 2.1 中的“scanf("%f",&b);”是要求将一个单精度类型的数据输入到变量 b 中,其中的"%f"是作为指示变量 b 为单精度类型。其一般格式为:

scanf("格式控制字符串",地址表列);

其中,格式控制字符串的作用与 printf 函数相同,但不能显示非格式字符串,也就是不能显示提示字符串。地址表列中给出各变量的地址。地址是由地址运算符“&”后跟变量名组成的,如:&a, &b 分别表示变量 a 和变量 b 的地址。

请看下例:

【例 2.8】

【程序】

```
main(){
 int a,b,c;
 printf("请输入 a,b,c\n");
 scanf("%d%d%d",&a,&b,&c);
 printf("a=%d,b=%d,c=%d",a,b,c);
}
```

scanf 函数本身不能显示提示串,所以先用 printf 语句在屏幕上输出提示,请用户输入 a、b、c 的值。当执行 scanf 语句时,进入用户屏幕等待用户输入。用户输入 7 8 9 后按下回车键,则程序执行 scanf()的下一语句。在 scanf 语句的“%d%d%d”控制三个整型变量,在输入时要用一个以上的空格或回车键作为每两个输入数之间的间隔。如:

```
7 8 9
```

或

```
7
8
9
```

scanf()函数的控制符与 printf()函数相仿,详见表 2.7:

表 2.7 scanf 函数常用格式控制符

格 式	字符意义
d	输入十进制整数
o	输入八进制整数
x	输入十六进制整数
u	输入无符号十进制整数
f 或 e	输入实型数(用小数形式或指数形式)

高清PDF原创 www.gqpdf.com
更多高清PDF图书 www.gqpdf.com

续表

格　式	字符意义
c	输入单个字符
s	输入字符串

使用 scanf 函数还必须注意以下几点：

(1)scanf 中要求给出变量地址，如只给出变量名则会出错。如“scanf("%d",a);”是非法的，应改为“scnaf("%d",&a);”才是合法的。

(2)在输入多个数值数据时，若格式控制串中没有非格式字符作输入数据之间的间隔则可用空格、TAB 或回车作间隔。C 编译在碰到空格、TAB、回车或非法数据(如对“%d”输入“12A”，A 即为非法数据)时即认为该数据结束。

(3)若在控制符之间用其他符号分格，输入时必须用这个符号分隔。如“scanf("%d#%d",&a,&b);”输入时应键入“12#30”，a，b 才能正确接收到 12 和 30。

(4)scanf 函数中没有精度控制，如：“scanf("%5.2f",&a);”是非法的。不能企图用此语句输入小数为 2 位的实数。

(5)在输入字符数据时，若格式控制串中无非格式字符，则认为所有输入的字符均为有效字符。

例如：

```
scanf("%c%c%c",&a,&b,&c);
```

输入为：

```
d e f
```

则把 'd' 赋予 a，' ' 赋予 b，'e' 赋予 c。
只有当输入为：

```
def
```

时，才能把 'd' 赋予 a，'e' 赋予 b，'f' 赋予 c。
如果在格式控制中加入空格作为间隔，如：

```
scanf("%c%c%c",&a,&b,&c);
```

则输入时各数据之间可加空格。

2.7　单个字符输入与输出函数

C 语言的库函数中还提供了对于单个字符输入和输出的函数。

putchar 函数是字符输出函数，其功能是在显示器上输出单个字符。其一般形式为：

putchar(字符变量);

如“putchar(a);”的作用等价于“printf("%c",a);”。

getchar 函数的功能是从键盘上输入一个字符。其一般形式为：

getchar();

如“x＝getchar();”等价于“scanf("%c",&x);”。

下面这个例子的作用是输入一个小写字母，输出它对应的大写字母。

【例 2.9】

【程序】

```
#include<stdio.h>
main()
{ char x;
printf("请输入一个小写字母:");
x=getchar();
x-=32;/*小写字母的ASCII码比大写字母多32*/
putchar(x);
}
```

2.8 本章小结

掌握 C 语言的基本语法和格式是程序设计的入门，各种算术运算符、简单数据类型、常量与变量、表达式、输入(scanf)与输出(printf)函数都是程序设计中最基础的元素，当然这一部分学习起来会有些枯燥，但只要我们能通过阅读程序和编写程序的方法将其不断巩固，就能很快入门。

课后练习

人高精品原创 www.gqpdf.com

1.写出以下程序运行的结果。

```
main( )
{ char c1='a',c2='b',c3='c',c4='\101',c5='116';
printf("a%c b%c c\tc%c c\tabc\n",c1,c2,c3);
printf("\t\b%c %c",c4,c5);
}
```

2.写出以下程序运行的结果。

```
main( )
{ int i,j,m,n; i=8; j=10;m=++i; n=j++;
printf("%d,%d,%d,%d",i,j,m,n);
}
```

3.设圆半径 $r=1.5$，圆柱高 $h=3$，求圆周长，圆面积，圆球表面积，圆球体积，圆柱体积。

更多高清PDF图书 www.gqpdf.com

用scanf输入数据,输出计算结果;输出时要求有文字说明,取小数点后2位数字,请编写程序。

4.编程序:用getchar函数读入两个字符给c1,c2,然后分别用putchar和printf函数输出这两个字符。并思考以下问题:(1)变量c1,c2应定义为字符型还是整型?(2)或两者皆可?

5.假设m是一个三位数,则写出将m的个位,十位,百位反序而成的三位数(例如:123反序为321)的C语言表达式。

6.已知"int x=10,y=12;"写出将x和y的值互相交换的表达式。

高清PDF原创 www.gqpdf.com

第 3 章

分支程序设计

高清PDF原创 www.gqpdf.com

【本章主要内容】

本章主要介绍 C 语言分支程序的设计，先介绍分支程序在实际的运用例子，再介绍 if 语句的语法和用法、关系运算符、逻辑运算符和条件运算符，分析 if...else 的各种情况，if 语句的嵌套使用，最后介绍 switch 语句的用法。

【本章要求】

(1)了解分支程序结构的概念。
(2)掌握 if 语句的语法和用法。
(3)掌握关系运算符、逻辑运算符及条件表达式的使用。
(4)学会写 if...else 语句的嵌套程序。
(5)了解 switch 语句的语法。

3.1 基本分支程序

3.1.1 分支程序例

【例 3.1】 为鼓励居民节约用水，自来水公司采取按月用水量分段计费的办法，若月用水量小于或等于 20 吨，则每吨收 1.45 元，超出 20 吨部分按每吨 3.5 元计费，试设计一程序，输入用户的用水量，计算出应收的水费。

【分析】 根据要求，我们可以设一变量 x 作为用户的用水量，再设一变量 y 作为应收的水费，则可以列出如下两条表达式：

(1) y=1.45 * x　　(当 x≤20)

更多高清PDF图书 www.gqpdf.com

(2) y=1.45*20+3.5*(x−20)(当 x>20)

【程序】

```
#include<stdio.h>
main()
{float x,y;
  printf("请输入用水吨数:");
  scanf("%f",&x);
  if(x<=20)  /*如果用水量小于或等于20吨*/
    y=1.45*x;
  else       /*否则*/
    y=1.45*20+3.5*(x−20);
  printf("应收水费%.2f元\n",y);
}
```

以上程序并不难懂,关键语句在“if(x<=20) y=1.45*x; else y=1.45*20+3.5*(x−20);”它表示如果 x 小于或等于 20 则按公式 y=1.45*x 计算水费,否则按 y=1.45*20+3.5*(x−20)计算水费。这就是 C 语言中的判断语句 if...else 语句。我们来看看它的一般语法。

3.1.2　if 语句的基本语法

语法:

if(表达式)

　语句 1;

else

　语句 2;

其执行过程是先判断表达式是否成立,如果成立则执行语句 1,否则(不成立)执行语句 2,其流程如图 3.1 所示。

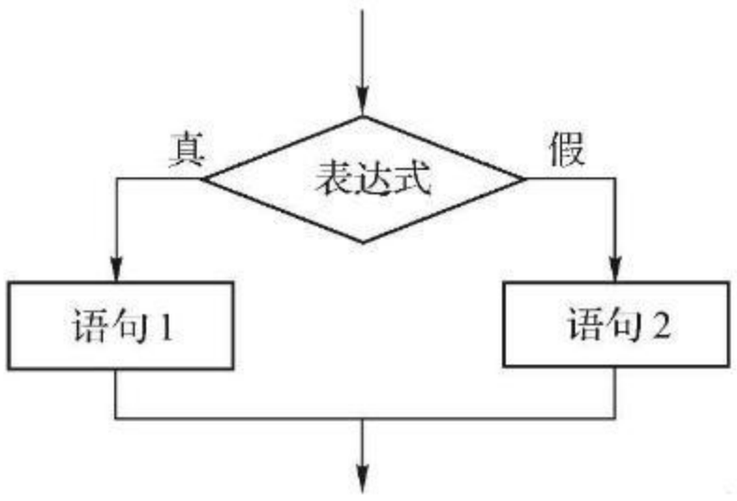


图 3.1　if...else 流程

即时练习

有一程序段如下:

```
#include<stdio.h>
main(    )
{int x,y;
scanf("%d",&x);
if(x<0)
    y=−x;
else
    y=x;
```

```
printf("y=%d",y);
}
```

若 x 的值为 5 则 y 的值为_____,若 x 的值为 -8 则 y 的值为_____。

3.2 关系运算符与逻辑运算符

3.2.1 关系运算符

在上面的程序中,if 语句后的“表达式”常常会用到关系运算符,如“<=”,那么 C 语言的关系运算符还有哪些呢? C 语言的关系运算符有 6 个,见表 3.1。

表 3.1 关系运算符

运算符	<	<=	>	>=	==	!=
名称	小于	小于或等于	大于	大于或等于	等于	不等于

要注意的事情是,C 语言的关系表达式的运算值是一个“逻辑值”,当表达式成立时,运算值为 1,而不成立时,其运算结果为 0,也就是说关系运算的结果不是 1 就是 0。如:

(1) d=3>2; 则 d 的值为 1。

(2) c=6==7+8 则 c 的值为 0。

即时练习

写出下面表达式的结果:

(1)56+8<=90 结果:

(2)45 * 3==5 * 43 结果:

3.2.2 逻辑运算符

C 语言的逻辑运算符有“&&、||、!”这三个,通常“&&”和“||”用来连接两个或多个条件,“!”用于将一个条件置反,我们先来看一个例子。

【例 3.2】 输入一件商品的单价及数量,如果单价和数量都大于 0 则打印出金额(=数量 * 单价),否则打印“输入数据有错误!”。

【分析】 先设两个变量 float x,y 分别表示“单价”和“数量”。当程序要求判断的条件有两个或以上时,往往需要用一些运算符来表示这些条件间的关系,在本例中可以理解成 x>0 和 y>0 这两个条件要同时成立。在 C 语言中用 &&(与)来表示两个条件同时成立。

【程序】

```
#include<stdio.h>
```

更多高清PDF图书 www.gqpdf.com

```
main()
{float x,y;
printf("请输入单价和数量:");
scanf("%f%f",&x,&y);
if(x>0 && y>0)
printf("金额:%.2f元\n",x*y);
else
printf("输入数据有错误!\n");
}
```

上面这个程序中用了 &&(与)运算符来连接两个条件,表示两个条件要同时成立,整个条件表达式才成立。其真值见表 3.2。

表 3.2　&&(与)的真值表

表达式 1	表达式 2	表达式 1 && 表达式 2
真	真	真
真	假	假
假	真	假
假	假	假

高清PDF原创 www.gaopdf.com

C 语言的关系运算符共有 3 个,除了 &&(与)之外还有||(或)和!(非),下面列出||和 !的真值表(见表 3.3 和表 3.4)。

表 3.3　||(或)的真值表

表达式 1	表达式 2	表达式 1 \|\| 表达式 2
真	真	真
真	假	真
假	真	真
假	假	假

表 3.4　!(非)的真值表

表达式 1	!表达式
真	假
假	真

通常,||(或)用于表示两个条件表达式中只要有一个为真,则整个表达式的值就为真,即表示两个条件中只要有一个成立,则整个表达式成立。例 3.2 的程序也可以改写成用||来连接两个条件。程序如下:

【程序】

```
#include<stdio.h>
main()
```

```
{float x,y;
  printf("请输入单价和数量:");
  scanf("%f%f",&x,&y);
  if(x<=0 || y<=0)
    printf("输入数据有错误!\n");
  else
    printf("金额:%.2f元\n",x*y);
}
```

请注意条件的变化:x<=0 || y<=0,程序可理解成当 x<=0 或者 y<=0 时,则打印“输出数据有错误!”,否则计算出金额。

逻辑运算符 !用于将条件置于相反,即若原表达式为真,加上 !后变为假,若原表达式为假,加上 !后变为真。

如:x=10<20,则 x 的值为 1,若有 x= !(10<20),则 x 的值为 0。

【注意事项】

在学习逻辑运算符时,还有几个事项是要注意的:

(1)C 语言在给出关系表达式或逻辑表达式计算的结果,当为真时,给出 1,当为假时给出 0;而在判断真假时,以非零为真,以零为假。如程序:

```
#include<stdio.h>
main()
{int x;
  x=-3;
 if(x)
   printf("OK\n");
 else
 printf("SORRY\n");
 }
```

其运行的结果为:OK。因为 if()中的表达式为非零,非零即为真。

(2)注意几种运算符的优先级,由高到低的顺序依次是:!(非)→算术运算符→关系运算符→&&(与)和||(或)→赋值运算符。请看下面的程序段:

```
int x=10,y=20,z;
z= !x+y>18&& x*3 == x+y;
```

这个表达式的运算次序为:

①先运算 !x,结果为 0(!的优先级最高),表达式可看成 z=0+y>18 && x*3 == x+y;

②其次计算算术运算符,即 0+y,x*3 和 x+y,结果分别是 20,30,30,表达式可看成 z=20>18 && 30 == 30;

③下一步是关系运算,即计算 20>18 和 30 == 30,结果分别 1 和 1,表达式可看成 z=1 && 1;

④再下来是逻辑运算符 &&,即计算 1 && 1 结果为 1;

更多高清PDF图书 www.gqpdf.com

⑤最后是赋值运算:z=1；也就是说最终z的结果为1。

(3)C语言的逻辑表达在进行逻辑运算时,如果通过前一部分表达式的值就能断定整个表达表的值,那么,后一部分的表达式则不再进行运算。如：

```
int x=4,y=8,z;
z=++x>y&&++y>x;
```

则该表达式运算完后,x的值为5,y的值为8而不是9,z的值为0,因为++x>y的结果为0,则无论&&右边的运算结果是多少,整个表达式的结果都为0,故系统不再对右边部分的表达式求解,也就是说++y>x无需运算,所以y的值仍为8。同理,

```
int x=3,y=1,z;
z=++x>y ||++y>x;
```

运算结果:x为4,y为1,z为1。

即时练习

写出下面表达式的结果,设a=3,b=4,c=5。

(1)x=a || b+c && b-c;　x的值为_____。

(2)x=!(b-a)>c || !(c-b == c/a);　x的值为_____。

(3)x=++a>b &&++c>a;　x的值为_____,a的值为_____,c的值为_____。

3.3　条件运算符

C语言有一个条件运算符,在某些情况下,用条件运算符会比用if...else语句简洁,如例3.1的程序可以改写成用条件运算符表示的程序,如下所示：

【程序】

```
#include<stdio.h>
main()
{float x,y;
printf("请输入用水吨数:");
scanf("%f",&x);
y=(x<=20)?1.45*x:1.45*20+3.5*(x-20);
printf("应收水费%.2f元\n",y);
}
```

程序中的条件运算符由?和:组成,它表示当x<=20为真时,整个表达式的值取:号前的表达1.45*x,否则,整个表达式的值取:号后的表达1.45*20+3.5*(x-20)。

由条件运算符组成条件表达式的一般形式为：

表达式 1 ? 表达式 2 :表达式 3

其求值规则为:如果表达式 1 的值为真,则以表达式 2 的值作为条件表达式的值,否则以表达式 3 的值作为整个条件表达式的值。条件表达式通常用于赋值语句之中。

即时练习

若 a=9,b=7

m=a/b > a-b ? a-b : a+b;

m 的值为:_____。

3.4 不带 else 的 if 语句

【例 3.3】 输入一个数,输出它的绝对值。

【程序】

```
#include<stdio.h>
main()
{float x;
printf("请输入一个数:");
scanf("%f",&x);
if(x<0)
  x=-x;
printf("它的绝对值是:%f\n",x);
}
```

程序中"if(x<0) x=-x;"表示如果 x<0 为真,则执行"x=-x;"否则不执行,而语句:"printf("它的绝对值是:%f\n",x);"是无论 x<0 成立与否都要执行的。这是一个有 if 而没有 else 的语句,是分支语句的一种形式。一般格式如下:

语法:

if(条件)

语句;

下一语句

其语义是:如果表达式的值为真,则执行其后的语句,否则不执行该语句,其流程如图 3.2 所示。

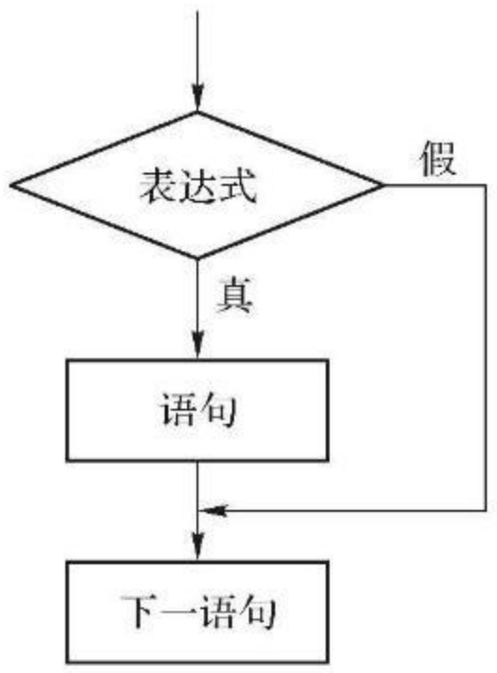


图 3.2 if 语句流程

即时练习

阅读下面程序:

更多高清PDF图书 www.gqpdf.com

```
#include<stdio.h>
main()
{char x;
x=getchar();
if(x>='a'&& x<='z')
  x=x-32;
printf("%c\n",x);
}
```

若从键盘输入'M',则输出:_____;若输入'f',则输出:_____。

【注意事项】

在if()之后不要随便加分号,这样会解释成为一个空语句,从而引起不必要的错误。如:

```
int x=9
if(x<0);
x=-x;
printf("x=%d\n",x);
```

输出的结果是-9。这是因为if(x<0)后加了一个分号,系统理解为空语句,也就是说如果x<0不成立,不被执行的是空语句,x=-x不受if(x<0)的约束,即无论条件成立与否,语句x=-x都会执行。

而下面程序则会产生语法错误:

```
if(x<=20);
y=1.45*x;
else
y=1.45*20+3.5*(x-20);
```

其原因是在if和else之间只能有一个语句,这里在if和else这间有两个语句,空语句和y=1.45*x。若在if和else之间需要多个语句则要用{}将它们合成一个复合语句,如:

```
if(x<=20)
{ y=1.45*x;
printf("应收水费:%.2f元\n",y);
}
else
{ y=1.45*20+3.5*(x-20);
  printf("应收水费:%.2f元\n",y);
}
```

高清PDF原创 www.gqpdf.com

3.5　if语句的嵌套使用

【例3.4】 已知某地计算个人所得税规则如下:

收入≤800　　所得税为 0；
800 ＜ 收入 ≤ 2000　　超出 800 部分按 5% 收；
2000 ＜ 收入 ≤ 8000　　超出 2000 部分按 10% 收；
8000 ＜ 收入　　超出 8000 部分按 20% 收；

试设计一程序，根据用户输入的个人收入，计算出应收税款。

【分析】

这个程序分了四种情况收取税款，我们仍用 if...else 来完成程序的设计，可以在 if 后的语句或是 else 后的语句再加入 if...else 语句。解题的程序可以有多种，在这里我们列举三个不同的程序来实现。

【程序 1】

```
include＜stdio.h＞
main()
{float x,y;
printf("请输入个人收入:");
scanf("%f",&x);
if(x＜＝800)              /*第 1层 if*/
  y＝0;
else                      /*第 1层 else*/
  {if(x＜＝2000)          /* 第 2层 if,此处为什么无需判断 x是否大于 800? */
    y＝(x－800)*0.05;
   else                   /*第 2层 else*/
    {if(x＜＝8000)        /*第 3层 if*/
      y＝1200*0.05＋(x－2000)*0.1
     else                 /*第 3层 else*/
      y＝1200*0.05＋6000*0.1＋(x－8000)*0.2;
    }
  }
printf("应收税款:%.2f\n",y);
}
```

上述程序中，在第一层 else 后的语句中嵌套着另一个 if...else 语句，在第二层 else 后的语句中也嵌套着另一个 if...else 语句。另外，请想一想为什么在第二层的 if 条件中无需判断 x 是否大于 800?

【程序 2】

```
#include＜stdio.h＞
main()
{float x,y;
printf("请输入个人收入:");
scanf("%f",&x);
if(x＜＝2000)             /*第 1层 if*/
  { if(x＜＝800)
```

更多高清PDF图书 www.gqpdf.com

```
y=0;
    else
y=(x-800)*0.05;
  }
else                          /*第1层else*/
  { if(x<=8000)
       y=1200*0.05+(x-2000)*0.1
    else
       y=1200*0.05+6000*0.1+(x-8000)*0.2;
  }
printf("应收税款:%.2f\n",y);
}
```

程序2在第一层的if后面的语句中和else后面的语句中各嵌套着一个if...else语句。这个例题也可以用下面的程序3来解决,但它的4个if语句是平行的,而不是嵌套的。

【程序3】

```
include<stdio.h>
main()
{float x,y;
printf("请输入个人收入:");
scanf("%f",&x);
if(x<=800)
  y=0;
if(x>800&& x<=2000)
  y=(x-800)*0.05;
if(x>2000&& x<=8000)
  y=1200*0.05+(x-2000)*0.1
if(x>8000)
  y=1200*0.05+6000*0.1+(x-8000)*0.2;
printf("应收税款:%.2f\n",y);
}
```

高清PDF原创 www.gqpdf.com

上面3个程序中,程序3的执行效率会稍低一些,因为无论x的值是什么,它的四个判断都会进行,而程序1和程序2有可能减少一些判断,如若x的值为500,则程序1只进行了一次if判断,程序2进行了2次,程序3进行了4次判断。但是程序3也有它的优点,这个程序的逻辑阅读起来似乎更清晰。

3.6　switch语句

C语言中可用于分支选择的还有一个语句 switch。我们先看一个用switch实现的程序。

【例 3.5】 输入一个 1～7 的数字,输出它所对应星期的英语单词。

【程序 1】

```
#include<stdio.h>
main()
{int a;
    printf("输入一个整数:    ");
    scanf("%d",&a);
    switch(a)
    {
        case 1:printf("Monday\n");break;
        case 2:printf("Tuesday\n");break;
        case 3:printf("Wednesday\n");break;
        case 4:printf("Thursday\n");break;
        case 5:printf("Friday\n");break;
        case 6:printf("Saturday\n");break;
        case 7:printf("Sunday\n");break;
        default:printf("输入错误\n");
    }
    printf("输出完毕\n");
}
```

【分析】 上面程序中 switch(a)对 a 的值进行判断,根据 a 的值,在后面对应的 case 找到其入口,若 a 的值与哪个 case 后的常量相等,则从该 case 处开始执行,如 a 的值为 5 则转入 case 5:printf("Friday\n")处执行。这里需要说明的有三点:

(1)case 后面必须是常量表达式,不能含变量。

(2)上例程序中的 default 不是必须的,它表示若所有 case 常量无一匹配,则执行 default后的语句。如上例中若 a 输入的值为 8,则没有一个 case 常量与之相匹配,则执行 default后的语句:printf("输入错误\n")。

(3)break 语句在此程序中的作用是跳出 switch 语句,在上例中若执行了 break,则直接跳到 printf("输出完毕\n")处执行。因为每一个 case 只是入口,当程序转入这个入口后,将从此处顺序执行下去。如我们将上例改为:

【程序 2】

```
#include<stdio.h>
main()
{int a;
    printf("输入一个整数:    ");
    scanf("%d",&a);
    switch(a)
    {
        case 1:printf("Monday\n");
        case 2:printf("Tuesday\n");
```

更多高清PDF图书 www.gqpdf.com

```
        case 3: printf("Wednesday\n");
        case 4: printf("Thursday\n");
        case 5: printf("Friday\n");
        case 6: printf("Saturday\n");
        case 7: printf("Sunday\n");
        default: printf("输入错误\n");
    }
    printf("输出完毕\n");
}
```

这样假设输入 a 的值为 5,则输出的结果为:

```
Friday
Saturday
Sunday
输入错误
输出完毕
```

在这里我们归纳 switch 语句的一般格式:

```
switch(表达式){
    case 常量表达式 1:  语句 1;
    case 常量表达式 2:  语句 2;
    …
    case 常量表达式 n:  语句 n;
    default          :  语句 n+1;
    }
```

其语义是:计算表达式的值。并逐个与其后的常量表达式值相比较,当表达式的值与某个常量表达式的值相等时,即执行其后的语句,然后不再进行判断,继续执行后面所有 case 后的语句。如表达式的值与所有 case 后的常量表达式均不相同时,则执行 default 后的语句。

高清PDF原创 www.gqpdf.com

即时练习

```
#include<stdio.h>
main()
{int x=7,y=1;
switch(x%3)
{case 0: y++;break;
  case 1: y++;
  case 2: y+=2;break;
  case 3: y+=3;
  }
printf("y=%d\n",y);
}
```

上面程序的运行结果是：_____。

3.7 本章小结

分支结构语句使得程序能够根据条件自行选择执行的语句，即使计算机能够思考。C 语言的分支语句主要有 if 和 switch 两个，if 是两个分支，switch 是多分支，但是 if 的嵌套使用使得 if 语句也能够实现多分支。switch 语句的语法格式与其他语言的多分支语句相比，较为呆板，初学者不太容易掌握。

课后练习

高清PDF原创 www.gqpdf.com

1.若运行时输入 100↙，写出以下程序的运行结果。

```
main( )
{ int a ;
 scanf("%d",&a);
 printf("%s",(a%2 !=0)?"No":"Yes");
 }
```

2.写出以下程序的运行结果。

```
main( )
{ int a=2,b=7,c=5;
 switch(a>0)
 { case 1:
     switch(b<0)
         { case 1: printf("@"); break ;
           case 2: printf("!"); break ; }
   case 0:
       switch(c==5)
         { case 1: pritnf("*") ; break ;
           case 2: printf("#") ; break ;
           default : printf("#") ; break ;
         }
   default : printf("&");
   }
   printf("\n");
}
```

3.编程输入整数 a 和 b，若大于 100，则输出百位以上的数字，否则输出两数之和。

4.有三个整数 a，b，c，由键盘输入，输出其中最大的数。

5.输入三个整数 x，y，z，请把这三个数由小到大输出。

更多高清PDF图书 www.gqpdf.com

6.给出一百分制成绩,要求输出成绩等级'A','B','C','D','E'。90分以上为'A',80—89分为'B',70—79分为'C',60—69分为'D',60分以下为'E'。

7.企业发放的奖金根据利润提成。利润(I)低于或等于10万元时,奖金可提10%;利润高于10万元,低于20万元时,低于10万元的部分按10%提成,高于10万元的部分,可提成7.5%;20万到40万之间时,高于20万元的部分,可提成5%;40万到60万之间时高于40万元的部分,可提成3%;60万到100万之间时,高于60万元的部分,可提成1.5%,高于100万元时,超过100万元的部分按1%提成,从键盘输入当月利润I,求应发放奖金总数。

高清PDF原创 www.gqpdf.com

第 4 章

循环程序设计

【本章主要内容】

本章主要介绍 C 语言循环程序的设计，先介绍循环程序在实际中的运用例子，主要介绍 while 语句的语法和用法。再介绍 do...while 语句、for 语句的用法，分析了 for 语句的各种特殊用法，然后介绍循环语句的嵌套使用，最后介绍 continue 和 break 语句的使用。

【本章要求】

(1)了解循环程序结构的概念。
(2)掌握 while 语句的语法和用法。
(3)掌握 do...while 语句的语法和用法。
(4)掌握 for 语句的语法和用法。
(5)学会编写循环嵌套的程序。
(6)了解 continue 与 break 语句的作用。

4.1 while 循环语句

4.1.1 while 程序例

【例 4.1】 某企业 2005 年的生产总值是 600 万元，据预测以后每年的增长率为 10%，请打印出以后 10 年每年的生产总值。

【分析】 根据题意，设一变量 x 记载年生产总值，初值为 600 万，以后每年在原基础上乘以 1.1 即为当年生产总值。

更多高清PDF图书 www.gqpdf.com

【程序】

```
#include<stdio.h>
main()
{float x;
 int year;
 x=600;
 year=2005;
while(year<=2015)
  { year++;
    x=x*1.1;
    printf("%d年的生产总值为%g万元\n",year,x);
  }
}
```

【运行结果】

```
2006年的生产总值为 660 万元
2007年的生产总值为 726 万元
2008年的生产总值为 798.6 万元
2009年的生产总值为 878.46 万元
2010年的生产总值为 966.306 万元
2011年的生产总值为 1062.94 万元
2012年的生产总值为 1169.23 万元
2013年的生产总值为 1286.15 万元
2014年的生产总值为 1414.77 万元
2015年的生产总值为 1556.25 万元
```

程序中 while(year<=2015)是一个循环语句，其后的复合语句{ year++;　　x=x*1.1; printf("%d 年的生产总值为%g 万元\n",year,x);}为循环体语句。其含义是：只要 year<=2015 成立，则不断地执行其后的循环体语句。其流程图如图 4.1 所示：

其执行的过程，year 的初值为 2005，year<=2015 成立，则执行循环体语句，year++; x=x*1.1; year 的值为 2006，x 的值为 660，打印出“2006 年的生产总值为 660 万元”，之后，再判断 year<=2015 是否成立，如此循环了 10 次，直到 year 的值变为 2016 时，while 后的条件不成立，循环结束，跳到循环体语句后执行。

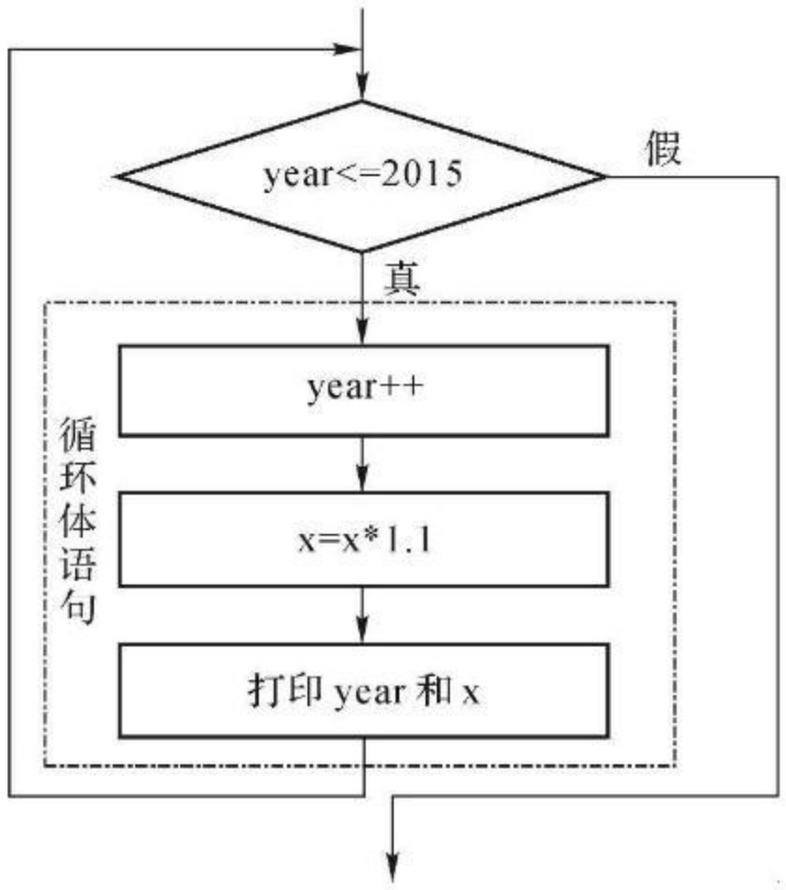


图 4.1　例 4.1 的流程

4.1.2 while 语法

while 语句的一般形式为：

while(表达式)

＜循环体语句＞

它的语义是：计算表达式的值，当值为真(非 0)时，执行循环体语句，直到表达式的值为假，则跳转到循环体语句之后。其流程如图 4.2 所示。

要注意的是，循环体语句仅为 while 之后的一个语句，如果循环体包含多个语句，则要用一对{}括起来，作为一个复合语句。

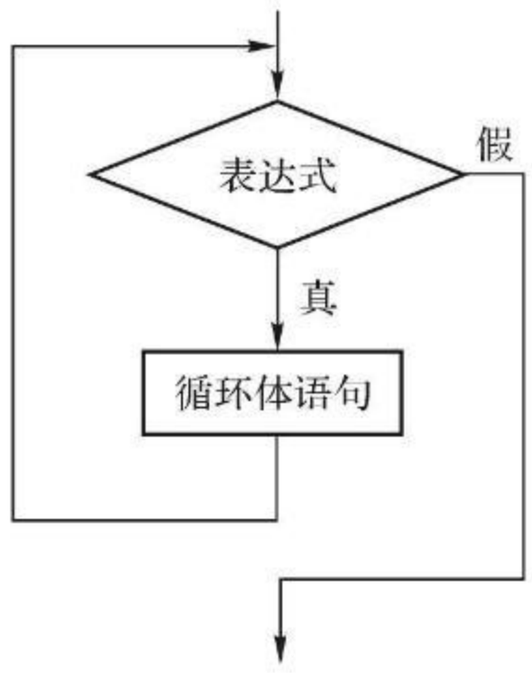


图 4.2 while 语句流程

下面请读者阅读两个例题，以加深对循环语句的认识。

【例 4.2】 从键盘输入一个整数，计算它的阶乘。

【程序】

```
#include<stdio.h>
main()
{int x,i;
 long y;
 i=1;y=1;
printf("请输入一个整数:");
scanf("%d",&x);
if(x>0)
  {while(i<=x)
      {y=y*i;
      i++;
      }
    printf("%d的阶乘是%ld\n",x,y);
  }
else
  printf("输入的数据错误\n");
}
```

【分析】 程序中 i 的初值为 1，故而循环语句“while(i<=x)”控制了循环体的执行次数为 x 次，每执行一次循环 i 的值增加 1，使得循环结束时 y 的值被赋为 1*2*3*……*x，也就是 x 的阶乘。

【例 4.3】 从键盘输入一行字符，统计其字符的个数。

【程序】

```
#include<stdio.h>
main(){
     int n=0;
     char c;
```

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

```
    printf("输入一行字符:\n");
    while((c=getchar())!='\n')
      n++;
    printf("共有%d个字符\n",n);
}
```

【分析】 本例程序中的循环条件为(c=getchar())!='\n',其意义是:从键盘输入的字符赋给变量c,只要c不等于回车符就继续循环。循环体n++完成对输入字符个数计数。从而程序实现了对输入一行字符的个数统计。

高清PDF原创 www.gqpdf.com

4.2　do...while循环语句

do...while语句与while语句非常相似,其语法如下:

do

<循环体语句>

while(表达式);

其流程如图4.3所示:

这个循环与while循环的不同在于:它先执行循环中的语句,然后再判断表达式是否为真,如果为真则继续循环;如果为假,则终止循环。因此,do...while循环至少要执行一次循环语句。

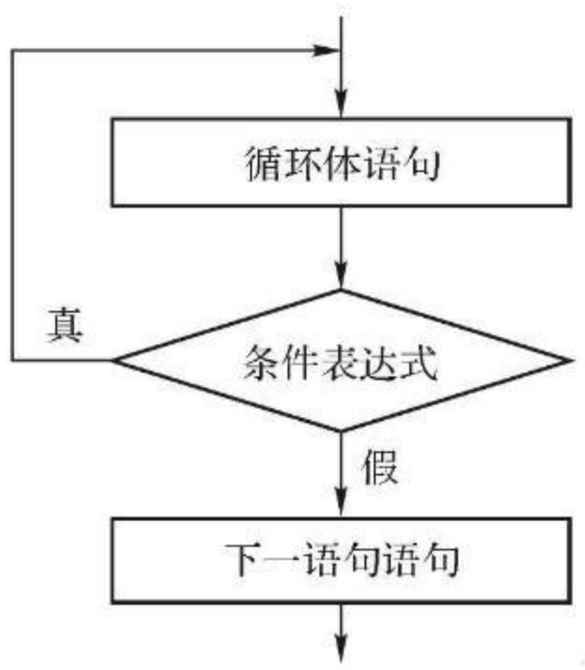


图4.3　do...while语句流程

【例4.4】 用do...while语句实现例4.1程序。

【程序】

```
#include<stdio.h>
main()
{float x;
 int year;
 x=600;
 year=2005;
 do
  {year++;
   x=x*1.1;
   printf("%d年的生产总值为%g万元\n",year,x);
  }while(year<=2015);
}
```

注意while与do...while的区别:

while:先判断后执行,循环体的执行次数可能为0;

do...while:先执行后判断,循环体的执行次数至少为1。

如下面程序：

```
#include<stdio.h>
main()
{int s,i;
 i=5; s=0;
while(i<=4)
{ s+=i;
 i++;
}
printf("s=%d\n",s);
}
```

其中循环体语句执行的次数为 0 次，执行结果为：s=0。

而下面程序：

```
#include<stdio.h>
main()
{int s,i;
 i=5; s=0;
while(i<=4)
{ s+=i;
 i++;
}
printf("s=%d\n",s);
}
```

其中循环体语句执行的次数为 1 次，执行结果为：s=5。

4.3 for 循环语句

C 语言中另一个循环语句是 for 语句，它是 C 语句中使用最多的循环语句。下面我们先来看看用 for 语句如何改写例 4.1

【例 4.5】 用 for 语句实现例 4.1 程序。

【程序】

```
#include<stdio.h>
main()
{float x;
 int year;
x=600;
 for(year=2005;year<=2015;year++)
   { x=x*1.1;
```

高清PDF原创 www.gqpdf.com
更多高清PDF图书 www.gqpdf.com

```
        printf("%d年的生产总值为%g万元\n",year,x);
    }
}
```

高清PDF原创 www.gqpdf.com

此程序中for后面的{}中的语句为循环体。循环的执行过程是:先执行year=2005;然后判断year<=2015是否为真,如果为真,则执行循环体语句{ x=x*1.1; printf("%d年的生产总值为%g万元\n",year,x);},循环体语句执行后再执行year++,再进行year<=2015的判断,如此循环10次,直到year<=2015为假,不再执行循环体,程序结束。

for语句的基本语法如下:

for(表达式1;表达式2;表达式3)

循环体语句

它的执行过程如下:

(1)先求解表达式1。

(2)再求解表达式2,若其值为真(非0),则执行for后面的循环体语句,然后执行下面第(3)步;若其值为假(0),则结束循环,转到第(5)步。

(3)求解表达式3。

(4)转回上面第(2)步继续执行。

(5)循环结束,执行for语句下面的一个语句。

其执行过程可用图4.4表示。

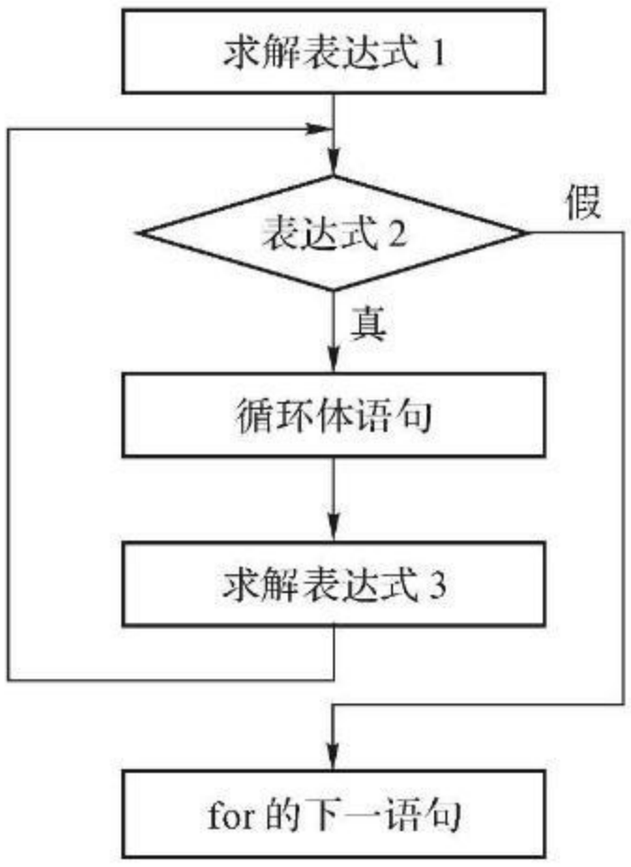


图4.4　for循环语句的流程

关于for语句的几点说明:

(1)通常表达式1可以是一个逗号表达式;如例4.5中,可以将x=600写在for的表达式1中:

```
for(year=2005,x=600;year<=2015;year++)
  { x=x*1.1;
    printf("%d年的生产总值为%g万元\n",year,x);
  }
```

(2)for 循环中的“表达式 1”、“表达式 2”和“表达式 3”可以缺省,但“;”不能缺省。

①例 4.5 缺省表达式 1 的写法:

```
#include<stdio.h>
main()
{float x;
 int year;
 x=600;  year=2005;
 for( ;year<=2015;year++)
   { x=x*1.1;
     printf("%d年的生产总值为%g万元\n",year,x);
   }
}
```

②例 4.5 缺省表达式 2 的写法:

```
#include<stdio.h>
main()
{float x;
 int year;
 x=600;
 for(year=2005;  ;year++)
   { if(year>2015) break;
     x=x*1.1;
     printf("%d年的生产总值为%g万元\n",year,x);

   }
}
```

程序中,缺省了表达式 2 则视为它的值一直为真,如果不在循环体中加入一个能跳出的语句,则会变成了死循环。所以程序中的 if(year>2015) break;就起到这个作用,在这里 break 的作用是跳出循环,执行循环体后的语句。

③例 4.5 缺省表达式 3 的写法:

```
#include<stdio.h>
main()
{float x;
 int year;
x=600;;
 for(year=2005;year<=2015;)
   { x=x*1.1;
     printf("%d年的生产总值为%g万元\n",year,x);
     year++;
   }
}
```

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

因为表达式3的执行次数与循环体语句的执行次数一样，所以要缺省表达式3，应将其相应的语句放在循环体内。有时for语句也会三个表达式同时省略，写成for(；；)。

4.4　循环语句的嵌套

【例4.6】 某工厂有3个车间，每个车间有4个班组，下班前每班组上报产量，统计总产量。

【分析】 工厂有3个车间，可以通过循环3次来读取数据，而每个车间有4个班组，又可在每个车间内设循环4次来读取数据，这样共读取12次数据，并进行累加。

【程序】

```
#include<stdio.h>
main()
{intx,i,j,s;
s=0;
for(i=1;i<=3;i++)     /*车间从1到3*/
  {for(j=1;j<=4;j++)  /*班组从1到4*/
    {printf("输入%d车间%d班组产量:",i,j);
     scanf("%d",&x);
     s+=x;            /*进行累加*/
    }
  }
printf("当天总产量:%d\n",s);
}
```

【运行结果】

```
输入1车间1班组产量:15
输入1车间2班组产量:20
输入1车间3班组产量:18
输入1车间4班组产量:21
输入2车间1班组产量:16
输入2车间2班组产量:16
输入2车间3班组产量:17
输入2车间4班组产量:19
输入3车间1班组产量:20
输入3车间2班组产量:15
输入3车间3班组产量:17
输入3车间4班组产量:16
当天总产量:210
```

这个程序共输入了12个数，因为内层循环体语句被执行了12次。我们不难看出，程序外

高清PDF原创 www.gqpdf.com

层的 for(i=1;i<=3;i++)语句(称为外循环)控制其循环体执行了 3 次,它的循环体中有另一个循环体语句 for(j=1;j<=4;j++)(称为内循环),它将被执行 3 次,而内层循环体每执行一次又将控制它自己的内循环体语句执行 4 次,所以内层循环体语句被执行了 12 次。

当然,循环的嵌套使用可以是由三种循环语句自由组合,即可是 for、while、do...while 任意组合,并没有什么限制。

下面我们再来看一个例子。

【例 4.7】 打印出下面的图形。

```
      *
    * * *
  * * * * *
* * * * * * *
```

【分析】 这个图形有 7 行,所以可以先设定一个 7 次的循环 for(i=1;i<=7;i++),再分析,第一行有 6 个空格,第 2 行有 5 个空格,……,可以得到空格数与行号的关系式是:空格数=7-行号;"*"个数与行号的关系是:"*"个数=2*行号-1。根据分析,我们利用一个两重循环程序来实现它。

【程序】

```
#include<stdio.h>
main()
{int i=1,j;
while(i<=7)/*外循环,控制 7行,也可用 for实现*/
  { for(j=1;j<=7-i;j++)/*打空格次数,也可用while或 do..while实现*/
      printf(" ");
    for(j=1;j<=2*i-1;j++)/*打印*号次数*/
      printf("*");
    printf("\n");
    i++;
  }
}
```

4.5 break 语句和 continue 语句

为了使循环控制更加灵活,C 语言提供了 break 语句和 continue 语句。break 语句强行结束循环,直接执行循环语句的下一语句。而 continue 语句是跳过循环体的余下语句部分,进行下一次循环。

更多高清PDF图书 www.gqpdf.com

4.5.1　break 语句

语法：

break

功能：强行结束循环，转向执行循环语句的下一条语句。

【例 4.8】 输入一个整数，判断它是否为素数。

【分析】 所谓素数即质数，是指仅能被 1 和它本身整除的整数。设此数为 x，我们可从 2 开始，一直到 x－1，每个数都拿来除 x，若期间发现有能整除的数，则 x 不是素数，否则是素数。

【程序】

```
#include<stdio.h>
main()
{int x,i;
printf("请输入一整数:");
scanf("%d",&x);
for(i=2;i<=x-1;i++)
  {if(x% i==0)
     break;
  }
if(i<x)
    printf("%d不是素数\n",x);
else
    printf("%d是素数\n",x);
}
```

程序中如果 i 在 2 到 x－1 间发现 if(x% i==0)成立，则跳出 for 循环，此时 i 的值将小于 x，据此判断对 x 不是素数，若 i==x，则说明 if(x% i==0)从未成立，则 x 为素数。为了减少程序判断的次数，通常也将循环写为 for(i=2;i<=x/2;i++)，或者是 for(i=2;i<=sqrt(x);i++)，这在数学上可以证明是可行的，这里就不作解释了。

要注意的是，在多重循环中，break 语句只跳出它所在的那一层循环。如下面程序：

【例 4.9】 打印出 100 到 200 之间的素数。

【程序】

```
#include<stdio.h>
main()
{int x,i;
  for(x=101;x<200;x+=2)
  { for(i=2;i<x;i++)
    {if(x% i==0)
       break;
    }
```

```
        if(i<x)
          printf("%5d",x);
      }
    }
```

4.5.2 continue 语句

语法:

continue

功能:对于 for 循环,跳过循环体其余语句,转向循环变量增量表达式(表达式 3)的计算;对于 while 和 do...while 循环,跳过循环体其余语句,转向循环继续条件的判定。

下面看两个例子:

【例 4.10】

【程序】

```
#include<stdio.h>
main()
{int i;
for(i=1;i<=10;i++)
  {if(i%2==0)
    continue;
   printf("%4d",i);
   }
}
```

【运行结果】

```
2  4  6  8  10
```

程序中,若 if(i%2==0)成立,则跳过后面的 printf("%4d",i)语句,直接执行 for 中的 i++。再看一例:

【例 4.11】

【程序】

```
#include<stdio.h>
main()
{int i=0;
while(i<10)
{i++;
 if(i%2==0)
   continue;
printf("%4d",i);
}
}
```

更多高清PDF图书 www.gqpdf.com

【运行结果】

```
2  4  6  8  10
```

程序中,若 if(i%2==0)成立,则跳过后面的 printf("%4d",i)语句,直接跳到 while(i<10)的判断。

4.6　本章小结

高清PDF原创 www.gqpdf.com

循环语句能轻松地控制使得需要重复执行的语句按指定的要求执行,当然,其作用远不止这个,许多算法的灵活实现都不可缺少循环语句。C 语言的循环语句主要有三种:while、for 和 do...while,其实 for 语句和 do...while 语句的功能均可由 while 语句实现,只是 for 用起来使得程序更简洁。本章关键的难点是要掌握循环的嵌套。

课后练习

1.输入一行字符,分别统计出其中英文字母,空格,数字和其他字符的个数。

2.一球从100米高度自由落下,每次落地后反跳回原高度的一半;再落下,求它在第10次落地时,共经过多少米?第10次反弹多高?

3.输入两个正整数 m 和 n,求其最大公约数和最小公倍数。

4.编程求1!+2!+3!+…+19!+20!。

5.猴子吃桃问题:猴子第一天摘下若干个桃子,当即吃了一半,还不瘾,又多吃了一个第二天早上又将剩下的桃子吃掉一半,又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第10天早上想再吃时,发现只剩下一个桃子了。求第一天共摘了多少。

6.判断101～200之间有多少个素数,并输出所有素数。

7.打印出如下图案(菱形)

```
    *
   * * *
  * * * * *
 * * * * * * *
  * * * * *
   * * *
    *
```

8.有一分数序列:2/1,3/2,5/3,8/5,13/8,21/13,...,求出这个数列的前20项之和。

第 5 章

函　数

【本章主要内容】

本章主要介绍函数的基本概念、函数的参数、函数的调用方法和变量的作用域。其中函数的基本概念主要介绍函数的一般定义方法；函数的参数主要讨论形式参数和实际参数的类型，声明和使用；函数的调用主要介绍函数的一般调用方法和递归调用方法；最后的变量作用域主要讨论变量的作用范围和生存时间。

【本章要求】

(1)了解函数的基本概念和函数的作用。
(2)理解函数的递归调用方法、函数参数的使用方法和变量的作用域。
(3)掌握函数的一般定义和调用方法，并能用函数编写简单的程序实现特定功能。

5.1　函数的基本概念

5.1.1　函数的引入

活动 1　了解函数的作用。

请读者根据所学的知识用两种不同的方法设计一个程序，求出两个整型数字的最大值并讨论这两种方法的优劣。

【例 5.1】 使用普通方法求出两数的最大值。

【分析】 读者运用所学的知识解此题时，只需在 main()函数里接收两个输入的数字，判断之后将较大值者打印出来即可。以下是实现该功能的代码：

更多高清PDF图书　www.gqpdf.com

【程序】

```
main()    /* 主函数 */
{
  int a, b, c;    /* 声明部分,定义变量 */
  scanf("%d,%d", &a, &b);    /* 输入变量 a 和 b 的值 */
  if(a > b)
      c=a;
  else
      c=b;
  printf("max=%d\n", c);    /* 输出 c 的值 */
}
```

这是一个很简单的程序,读者可能觉得用三目运算符 c=a > b ? a : b 直接就可以把题目解答出来,这里我们不去讨论算法的效率问题,而是考虑有没有另一种更好的解决方法使我们程序更加健壮、可读性更好?

【例 5.2】 使用函数调用求出两数的最大值。

【分析】 我们尝试用函数调用的方法来解这道题,源代码如下所示:

【程序】

```
intmax(intx, inty)          /* 定义 max 函数,函数值为整型,形式参数 x,y 为整型 */
{
  int z;                    /* max 函数中的声明部分,定义本函数中用到的变量 z 为整型 */
  if(x > y)
      z=x;
  else
      z=y;
  return z;                 /* 将 z 的值返回,通过 max 带回调用处 */
}
main()                      /* 主函数 */
{
  int a, b, c;              /* 声明部分,定义变量 */
  scanf("%d,%d", &a, &b);   /* 输入变量 a 和 b 的值 */
  c=max(a, b);              /* 调用 max 函数,将得到的值赋给 c */
  printf("max=%d\n", c);    /* 输出 c 的值 */
}
```

我们可以看到这段程序除了 main 函数之外还有另外一个函数 int max(int x, int y)。研究该函数的代码我们发现该函数的功能便是得到两个数的最大值并返回(这里是通过变量 z 来返回最大值)。程序执行的顺序是先执行 main 函数,在主函数里首先输入两个数,然后通过主函数调用 max 函数得到最大值赋给 c 并输出。

读者可能觉得这段代码比例 5.1 的代码要长,解决问题的方法不简洁。但是大家仔细观察主函数就会发现主函数十分简洁,解决问题的关键只有一条语句。在实际应用中,问题可能是复杂的并且有可能是由多个子问题组成的,如果按照例 5.1 的写法将使主函数变得

难以阅读和理解;如果按照例 5 .2 的写法解决问题,则不论问题多复杂,主函数都显得条理清晰。不仅如此,如果下次我们在另外的地方需要用到这个功能时,只需调用这个已经写好的函数便可,这是符合软件工程代码可重用的原则的。函数就像我们家里放东西的一个个抽屉,每个抽屉里面放不同种类的东西,我们需要这些东西的时候就去抽屉里面拿(即调用函数)。下面我们给出函数的概念。

5 .1 .2 函数的概念

函数就是将实现特定功能的代码封装在一起进行命名,可实现多次调用的子程序。

我们在接触函数的概念之前其实已经使用过各种各样的函数,比如 printf、scanf 这两个最常用的函数,主函数 main 本身就是一个函数。由此我们发现,一个 C 语言程序至少应该包含一个函数,即最少要有主函数。主函数可以调用其他函数,而不允许被其他函数调用。因此,C 程序的执行总是从 main 函数开始,完成对其他函数的调用后再返回到 main 函数,最后由 main 函数结束整个程序。一个 C 源程序必须有,也只能有一个主函数 main。

在实际的应用中,程序往往由多个函数组成。函数是 C 源程序的基本模块,用户可把自己的算法编成一个个相对独立的函数模块,然后用调用的方法来使用函数。可以说 C 程序的全部工作都是由各式各样的函数完成的,所以也把 C 语言称为函数式语言。

由于采用了函数模块式的结构,C 语言易于实现结构化程序设计。使程序的层次结构清晰,便于程序的编写、阅读、调试。

5 .1 .3 函数的分类

在 C 语言中可从不同的角度对函数分类。

(1)从函数定义的角度看,函数可分为库函数和用户定义函数两种。

①库函数:由 C 系统提供,用户无须定义,也不必在程序中作类型说明,只需在程序前包含有该函数原型的头文件即可在程序中直接调用。在前面各章的例题中反复用到 printf、scanf、getchar、putchar、gets、puts、strcat 等函数均属此类。

②用户定义函数:由用户按需要写的函数。对于用户自定义函数,不仅要在程序中定义函数本身,而且在主调函数模块中还必须对该被调函数进行类型说明,然后才能使用。

(2)C 语言的函数兼有其他语言中的函数和过程两种功能,从这个角度看,又可把函数分为有返回值函数和无返回值函数两种。

①有返回值函数:此类函数被调用执行完后将向调用者返回一个执行结果,称为函数返回值。如数学函数即属于此类函数。由用户定义的这种要返回函数值的函数,必须在函数定义和函数说明中明确返回值的类型。

②无返回值函数:此类函数用于完成某项特定的处理任务,执行完成后不向调用者返回函数值。这类函数类似于其他语言的过程。由于函数无须返回值,用户在定义此类函数时可指定它的返回为“空类型”,空类型的说明符为“void”。

(3)从主调函数和被调函数之间数据传送的角度看又可分为无参函数和有参函数两种。

更多高清PDF图书 www.gqpdf.com

①无参函数:函数定义、函数说明及函数调用中均不带参数。主调函数和被调函数之间不进行参数传送。此类函数通常用来完成一组指定的功能,可以返回或不返回函数值。

②有参函数:也称为带参函数。在函数定义及函数说明时都有参数,称为形式参数(简称为形参)。在函数调用时也必须给出参数,称为实际参数(简称为实参)。进行函数调用时,主调函数将把实参的值传送给形参,供被调函数使用。

(4)C 语言提供了极为丰富的库函数,这些库函数又可从功能角度作以下分类。

①字符类型分类函数:用于对字符按 ASCII 码分类:字母,数字,控制字符,分隔符,大小写字母等。

②转换函数:用于字符或字符串的转换;在字符量和各类数字量(整型,实型等)之间进行转换;在大、小写之间进行转换。

③目录路径函数:用于文件目录和路径操作。

④诊断函数:用于内部错误检测。

⑤图形函数:用于屏幕管理和各种图形功能。

⑥输入输出函数:用于完成输入输出功能。

⑦接口函数:用于与 DOS,BIOS 和硬件的接口。

⑧字符串函数:用于字符串操作和处理。

⑨内存管理函数:用于内存管理。

⑩数学函数:用于数学函数计算。

⑪日期和时间函数:用于日期,时间转换操作。

⑫进程控制函数:用于进程管理和控制。

⑬其他函数:用于其他各种功能。

以上各类函数不仅数量多,而且有的还需要硬件知识才能使用,因此要想全部掌握则需要一个较长的学习过程。应首先掌握一些最基本、最常用的函数,再逐步深入。由于课时关系,我们只介绍了很少一部分库函数,其余部分读者可根据需要查阅有关手册。

5.1.4　函数的定义



活动 2　观察函数的定义。

请读者观察例 5.2 里 max 函数的定义方法,然后思考以下几个问题:

①max 函数如何得到 a,b 两个数的值?

②max 函数如何将 a 和 b 的比较结果告诉 main 函数?

③max 函数传给 main 函数的数据是什么类型的?

通过观察例 5.2,我们发现 max 函数定义的头部 int max(int x, int y)和 main 函数中调用的形式 max(a, b)在结构上有很相似的地方,只不过一个有类型的说明,另一个无类型的说明。我们发现 max 函数在做比较的时候是拿 x 和 y 在作比较,而不是直接拿 a 和 b 在作比较,并将比较结果存放在变量 z 中,函数体最后一句话为"return z;",这应该就是函数 max 把比较结果传回给 main 函数的途径。最后我们发现函数定义的头部最开始写着 int,而调用时将函数赋值给 c 这个变量,这个变量也是整型。

综上所述并结合函数的定义形式和调用形式,我们得出的结论是 max 函数通过被 main 函数调用,在调用时通过括号里的 a 和 b 两个参数得到待比较两个数的值,在函数里面进行比较后通过 z 这个变量返回比较结果,比较结果的数据类型是整型。

通过以上活动我们初步了解了函数的定义形式,那是不是所有的函数都必须有初始值给它进行计算呢? 我们结合函数的概念就会明白,函数是实现特定功能的代码的集合,也就是说实现某些功能并不需要初始的参数,由此我们给出函数定义的一般形式。

1.无参函数定义的一般形式

无参函数的定义如下:

类型标识符 函数名()

{

声明部分

语句

}

其中类型标识符和函数名称为函数头。类型标识符指明了本函数的类型,函数的类型实际上是函数返回值的类型。该类型标识符与前面介绍的各种说明符相同。函数名是由用户定义的标识符,函数名后有一个空括号,其中无参数,但括号不可少。{}中的内容称为函数体。在函数体中声明部分,是对函数体内部所用到的变量的类型说明。

在很多情况下都不要求无参函数有返回值,此时函数类型符可以写为 void。

例如,输出"Hello,world"的函数定义如下:

有参函数的定义如下:

```
voidHello()
{
  printf("Hello,world \n");
}
```

这里,Hello 函数是一个无参函数,当被其他函数调用时,输出 Hello world 字符串。

2.有参函数定义的一般形式

有参函数的定义如下:

类型标识符 函数名(形式参数表列)

{

声明部分

语句

}

有参函数比无参函数多了一个内容,即形式参数表列。在形式参数表列中给出的参数称为形式参数,它们可以是各种类型的变量,各参数之间用逗号间隔。在进行函数调用时,主调函数将赋予这些形式参数实际的值,如例 5.2 中的 max(a,b),这里 a 和 b 我们称为实际参数。形参既然是变量,则必须在形参表中给出形参的类型说明。

例如,定义一个函数,用于求两个数中的大数,可写为:

更多高清PDF图书 www.gqpdf.com

```
intmax(int a, int b)
{
  if(a>b) return a;
  else return b;
}
```

第一行说明 max 函数是一个整型函数,其返回的函数值是一个整数。形参 a、b 均为整型量。a、b 的具体值是由主调函数在调用时传送过来的。在{}中的函数体内,除形参外没有使用其他变量,因此只有语句而没有声明部分。在 max 函数体中的 return 语句是把 a(或 b)的值作为函数的值返回给主调函数。有返回值的函数中至少应有一个 return 语句。

即时练习

编写一函数,求三个数的最大值并打印到屏幕上。

3.小结

函数定义是使用函数的第一个阶段,在定义函数时函数之间是互相独立的,一个函数并不从属于另一个函数,即函数不能嵌套定义,也可理解为所有函数都是平行的。

在开发软件的初始阶段,常常使用空函数来预先定义需要的功能,空函数即函数体{}中无任何内容的函数。程序员可以在将来准备扩充功能的地方写上一个空函数(函数名取将来采用的实际函数名),先占一个位置,以后用一个编好的函数代替它。这样做使程序的结构清楚,可读性好,以后扩充新功能方便,对程序结构影响不大。空函数在程序设计中常常是有用的。

在软件工程中,大的程序一般不全放在一个文件中,而将函数和其他内容(如预定义)分别放在若干个源文件中,再由若干源文件组成一个 C 程序。这样有利于软件开发的模块化,提高软件的可重用性。

5.2　函数参数的讨论

5.2.1　形式参数和实际参数

通过活动 2 的观察和 5.1 节的学习,读者已经初步了解了函数的参数,函数的参数分为形参和实参两种。在本小节中,进一步介绍形参、实参的特点和两者的关系。形参出现在函数定义中,在整个函数体内都可以使用,离开该函数则不能使用。实参出现在主调函数中,进入被调函数后,实参变量也不能使用。形参和实参的功能是作数据传送。发生函数调用时,主调函数把实参的值传送给被调函数的形参从而实现主调函数向被调函数的数据传送。

高清PDF原创 www.gqpdf.com

活动 3　观察函数形参和实参的关系和作用。

编写一程序求$\sum n_i$(即 1+2+3+…+n)的值。要求程序用函数实现,在编写的过程中思考如下问题:

(1)主函数如何得到 n 的值?

(2)n 的值在被调用函数(即实现累加的函数)里面有没有被改变?

(3)如果被调用函数的形参为 n,主函数的实参为 n,那么被调用函数在做累加的时候是拿形参做累加还是拿实参做累加?

【例 5.3】 求$\sum n_i$(即 1+2+3+…+n)的值。

【分析】 可定义一个函数 void s(int n)实现此功能。

【程序】

```
void s(int n)
{
   int i;
   for(i=n-1;i>=1;i--)
       n=n+ i;
   printf("n=%d\n",n);
}
main()
{
   int n;
   printf("input number\n");
   scanf("%d",&n);
   s(n);
   printf("n=%d\n",n);
}
```

本题在主函数中输入 n 值,并作为实参,在调用时传送给 s 函数的形参量 n(注意,本例的形参变量和实参变量的标识符都为 n,但这是两个不同的量,各自的作用域不同)。在主函数中用 printf 语句输出一次 n 值,这个 n 值是实参 n 的值。在函数 s 中也用 printf 语句输出了一次 n 值,这个 n 值是形参最后取得的 n 值 0。从运行情况看,输入 n 值为 100。即实参 n 的值为 100。把此值传给函数 s 时,形参 n 的初值也为 100,在执行函数过程中,形参 n 的值变为 5050。返回主函数之后,输出实参 n 的值仍为 100。可见实参的值不随形参的变化而变化。

由此我们进一步总结函数的形参和实参的特点:

(1)形参变量只有在被调用时才分配内存单元,在调用结束时,即刻释放所分配的内存单元。因此,形参只有在函数内部有效。函数调用结束返回主调函数后则不能再使用该形参变量,即在调用结束后,形参所占的内存单元也被释放。

(2)实参可以是常量、变量、表达式、函数等,无论实参是何种类型的量,在进行函数调用时,它们都必须具有确定的值,以便把这些值传送给形参。因此应预先用赋值、输入等办法使实参获得确定值。

更多高清PDF图书 www.gqpdf.com

(3)实参和形参在数量上,类型上,顺序上应严格一致,否则会发生类型不匹配的错误。

(4)函数调用中发生的数据传送是单向的。即只能把实参的值传送给形参,而不能把形参的值反向地传送给实参。因此在函数调用过程中,形参的值发生改变,而实参中的值不会变化。在内存中,实参单元与形参单元是不同的存储单元。

即时练习

编写一函数,取得由 main 函数传来的两个值,将第一个数作为第二个数的乘方进行计算并将结果打印到屏幕上。

5.2.2　函数的返回值

活动 4　讨论函数的返回值。

在活动 3 中我们通过一个无返回值的函数实现计算 1 到 n 累加值的功能。请读者仔细研究例 5.3 并思考以下问题:

(1)除了在累加函数中将累加值输出之外,有没有其他方法将累加值输出?

(2)如果累加值由累加函数返回,则此返回值应为什么类型?

(3)如果在主函数中定义一个 float 型的变量 count 并将累加函数的返回值赋给 count,假设累加函数的返回值为 int,那么 count 的类型是否会随函数返回值而改变?

【例 5.4】 使用函数返回值求$\sum n$(即 1+2+3+…+n)的值。

【程序】

```
int s(int n)
{
  int i;
  for(i=n-1;i>=1;i--)
      n=n+ i;
  return n;
}
main()
{
  int n;
  int count;
  printf("input number\n");
  scanf("%d",&n);
  count= s(n);
  printf("n=%d\n",count);
}
```

在例 5.4 中我们使用函数返回值的方法得到累加和,显然,这个值应该定义为整型。在 main 函数中我们定义了一个整型的变量来存放函数的返回值,假设输入的 n 值为 12,则程

序的执行结果如下：

【运行结果】

```
input number
12
n=78
```

请读者思考活动 4 中的第三个问题，如果我们将 count 的类型变为浮点型那运行的结果将会发生怎样的变化呢？由前面学过的知识我们知道一个整型的数字为一个浮点型的变量赋值，结果应该是变量的类型不变，那么函数给变量赋值是否遵循这一原则呢？

【例 5.5】 函数返回值的进一步讨论。

【程序】

```
int s(int n)
{
  int i;
  for(i=n-1;i>=1;i--)
      n=n+i;
  return n;
}
main()
{
  int n;
  float count;
  printf("input number\n");
  scanf("%d",&n);
  count=s(n);
  printf("n=%f\n",count);
}
```

例 5.5 是在例 5.4 的基础上对主函数进行更改，累加函数不变。假设输入的 n 值为 12，则程序的执行结果如下：

【运行结果】

```
input number
12
n=78.000000
```

由此我们发现，如果函数返回值的类型与被赋值变量的类型不同时，会发生隐式的类型转换，即函数的返回值类型转换为变量的类型。

最后我们来思考一下，如果返回值类型与函数类型不同，则函数应如何处理？

【例 5.6】 返回值类型与函数类型不同。

【程序】

```
intmax(floatx, floaty)
{
```

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

```
    float z;
    z=x>y?x:y;
    return(z);
}
voidmain()
{
    float a,b;
    int c;
    scanf("%f%f",&a,&b);
    c=max(a,b);
    printf("Max is%d\n",c);
}
```

假设输入是2.5和1.5,则程序的运行结果如下:

【运行结果】

```
2.5,1.5
Max is 2
```

从运行结果我们发现,两个浮点数的较大者2.5被转化成了整型2输出,这说明如果返回值类型与函数类型不同,则以函数类型为准,在函数内部自动进行类型转换。

通过活动4的开展和例5.6的学习,我们可以进一步明确函数返回值的概念和特点。

函数的返回值也称函数的值,是指函数被调用之后,执行函数体中的程序段所取得的并返回给主调函数的值。如调用正弦函数取正弦值,调用例5.4的s函数取得的累加值等。对函数的值(或称函数返回值)有以下一些说明:

(1)函数的值只能通过return语句返回主调函数。

return　语句的一般形式为:

return 表达式;

或者为:

return(表达式);

该语句的功能是计算表达式的值,并返回给主调函数。在函数中允许有多个return语句,但每次调用只能有一个return语句被执行,因此只能返回一个函数值。

(2)函数值的类型和函数定义中函数的类型应保持一致。如果两者不一致,则以函数类型为准,在函数内部自动进行类型转换。如果将函数的返回值赋给主调函数中的一个变量,此变量的类型与函数返回值不同则将以变量的类型为准进行隐式的类型转换。

(3)如函数值为整型,在函数定义时可以省去类型说明。

(4)不返回函数值的函数,可以明确定义为“空类型”,类型说明符为“void”。如例5.3中函数s并不向主函数返函数值,因此可定义为:

```
void s(int n)
{
  ......
}
```

一旦函数被定义为空类型后，就不能在主调函数中使用被调函数的函数值了。例如，在定义 s 为空类型后，在主函数中写入下述语句：

```
count=s(n);
```

就是错误的。

为了使程序有良好的可读性并减少出错，凡不要求返回值的函数都应定义为空类型。

即时练习

编写一函数，取得由 main 函数传来的两个值，将第一个数作为第二个数的乘方进行计算并将结果传回给 main 函数，由 main 函数打印到屏幕上。

高清PDF版 www.gqpdf.com

5.2.3 函数参数与带参宏的比较

我们前面已经介绍过宏的概念和无参宏的使用方法，本小节介绍带参宏的用法并将其与函数作比较。C 语言允许宏带有参数。在宏定义中的参数称为形式参数，在宏调用中的参数称为实际参数。

对带参数的宏，在调用中，不仅要将宏展开，而且要用实参去代换形参。

带参宏定义的一般形式为：

#define 宏名(形参表) 字符串

在字符串中含有各个形参。

带参宏调用的一般形式为：

宏名(实参表);

例如：

```
#define M(y) y*y+3*y          /*宏定义*/
.....
k=M(5);                       /*宏调用*/
.....
```

在宏调用时，用实参 5 去代替形参 y，经预处理宏展开后的语句为：

```
k=5*5+3*5
```

【例 5.7】 了解带参宏的用法。

【程序】

```
#defineMAX(a,b) (a>b)? a:b
main()
{
   int x,y,max;
   printf("input two numbers:");
   scanf("%d,%d",&x,&y);
   max=MAX(x,y);
```

更多高清PDF图书 www.gqpdf.com

```
    printf("max=%d\n",max);
}
```

【分析】 上例程序的第1行进行带参宏定义,用宏名MAX表示条件表达式(a>b)? a:b,形参a,b均出现在条件表达式中。程序第7行max=MAX(x,y)为宏调用,实参x,y将替换形参a,b。宏展开后该语句为:

```
max=(x>y)? x:y;
```

用于计算x,y中的大数。

对于带参的宏定义有以下问题需要说明:

(1)带参宏定义中,宏名和形参表之间不能有空格出现。

例如把:

```
#define MAX(a,b)  (a>b)? a:b
```

写为:

```
#define MAX  (a,b)  (a>b)? a:b
```

将被认为是无参宏定义,宏名MAX代表字符串(a,b)(a>b)? a:b。宏展开时,宏调用语句:

```
max=MAX(x,y);
```

将变为:

```
max=(a,b)(a>b)? a:b(x,y);
```

这显然是错误的。

(2)在带参宏定义中,形式参数不分配内存单元,因此不必作类型定义。而宏调用中的实参有具体的值。要用它们去替换形参,因此必须作类型说明。这是与函数中的情况不同的。在函数中,形参和实参是两个不同的量,各有自己的作用域,调用时要把实参值赋予形参,进行"值传递"。而在带参宏中,只是符号代换,不存在值传递的问题。

(3)在宏定义中的形参是标识符,而宏调用中的实参可以是表达式。



活动5　比较带参的宏和带参函数两者的区别。

阅读例5.8和例5.9并思考如下问题:

(1)两个例子的运行结果一样吗?

(2)分析两个例子的运行流程?

(3)带参的宏和带参函数两者有什么本质的区别?

【例5.8】 带参的宏和带参函数的比较(使用带参函数)

【程序】

```
SQ(int y)
{
  return((y)*(y));
```

```
}
main()
{
   int i=1;
   while(i<=5)
       printf("%d\n",SQ(i++));
}
```

【例 5.9】 带参的宏和带参函数的比较(使用带参的宏)

【程序】

```
#define SQ(y) ((y)*(y))
main(){
 int i=1;
while(i<=5)
   printf("%d\n",SQ(i++));
}
```

通过阅读并运行程序,我们发现两个程序的运行结果是不同的,例 5.8 的运行结果为:

```
1
4
9
16
25
```

例 5.9 的运行结果为:

```
1
9
25
```

下面我们分析程序的运行流程:在例 5.8 中,函数调用是把实参 i 值传给形参 y 后自增 1;然后输出函数值,因而要循环 5 次;输出 1～5 的平方值。而在例 5.9 中宏调用时,只作代换。SQ(i++)被代换为((i++)*(i++))。在第一次循环时,由于 i 等于 1,其计算过程为:先将表达式(i++)*(i++)计算出来,这里是 i++,所以计算(i++)*(i++)相当于计算(i)*(i),结果为 1,然后表达式中前一个 i 自增 1 变为 2,表达式中第 2 个 i 自增 1 变为 3,因为宏只作代换,因此这里实际上是自增了两次。在第二次循环时,i 值已有初值为 3,先将表达式(i++)*(i++)计算出来结果为 9,然后表达式中前一个 i 自增 1 变为 4,表达式中第 2 个 i 自增 1 变为 5。进入第三次循环,由于 i 值已为 5,所以这将是最后一次循环。计算表达式的值为 5*5 等于 25。i 值再自增 1 变为 6,不再满足循环条件,停止循环。

由以上分析我们可以知道带参宏和带参函数有着本质的区别:

(1)函数调用时,先求出实参表达式的值,然后代入形参。而使用带参的宏只是进行简单的字符替换。

(2)函数调用是在程序运行时处理的,分配临时的内存单元。而宏展开则是在编译时进

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

行的,在展开时并不分配内存单元,不进行值的传递处理,也没有“返回值”的概念。

(3)对函数中的实参和形参都要定义类型,二者的类型要求一致,如不一致,应进行类型转换。而宏不存在类型问题,宏名无类型,它的参数也无类型,只是一个符号代表,展开时代入指定的字符即可。宏定义时,字符串可以是任何类型的数据。

(4)调用函数只可得到一个返回值,而用宏可以设法得到几个结果。

(5)宏替换不占运行时间,只占编译时间。而函数调用则占运行时间(分配单元、保留现场、值传递、返回)。

对于简短的表达式我们一般使用宏来代替,但是对于易产生歧义或者是复杂的问题,我们建议读者采用函数来解决,这也符合软件模块的重用性原则,使代码易于理解,可读性强,方便日后的使用和维护。

即时练习

编写程序求(R!+T!)/(M!-N!),其中,R、T、M、N均从键盘输入。要求用两种方法实现(即普通函数实现与带参宏实现)。提示:可编一求阶乘函数。

高清PDF原创 www.gaopdf.com

5.3　函数的调用

我们前面已经了解到在C语言程序中是通过对函数的调用来执行函数体的。C语言中,函数调用的一般形式为:

函数名(实际参数表)

对无参函数调用时则无实际参数表,但不能省略圆括号。实际参数表中的参数可以是常量,变量或其他构造类型数据及表达式,各实参之间用逗号分隔。实参与形参的个数应相等,类型和顺序应一致,他们之间是一一映射的关系。

5.3.1　函数调用的方式

活动6　讨论函数调用的几种常见方式。

观察例5.2和例5.3两个程序并思考以下问题:

(1)两个例子中的主函数调用其他函数的方式有什么不同?

(2)除了这两种函数调用方式之外还有其他的函数调用能够方式吗?

(3)能否只对例5.2的主函数进行简单的修改使其能求三个数的最大值?

通过对两个例子的观察,我们发现它们对函数的调用方式存在差别,在例5.2中,主函数通过语句“c=max(a,b);”调用max函数,这时max(a,b)函数把它的返回值赋给c,这是一条包含函数表达式的语句。而在例5.3中,主函数通过“s(n);”调用s函数,这时整个函数作为一条语句使用并完成某项功能。如果要求三个数的最大值,我们是否可以考虑先

将两个数作比较之后拿较大的数跟第三个数比较呢？如果可以，那么 max 函数将作为函数的参数来使用，即写成 max(c,max(a,b))的形式，这里拿 a 和 b 的较大者和 c 比较得出三者的最大值，我们可以参考例 5.10 体会这种解法。

【例 5.10】 函数作为函数的参数使用

【程序】

```
main()
{
  int a, b, c, r;
  scanf("%d,%d,%d",&a,&b,&c);
  r=max(c,max(a, b));
  printf("max=%d\n", r);
}
```

【分析】 在例 5.10 中，我们只是对例 5.2 的主函数进行了简单的修改而并不去修改 max 函数便实现了找出三个数最大者的功能，由此我们可以总结函数调用的三种方式：

(1)函数表达式：函数作为表达式中的一项出现在表达式中，以函数返回值参与表达式的运算。这种方式要求函数是有返回值的。例如：z=max(x,y)是一个赋值表达式，把 max 的返回值赋予变量 z。

(2)函数语句：函数调用的一般形式加上分号即构成函数语句，这时不要求函数返回值，只要求函数完成一定的操作。例如：printf ("%d",a);和 s(n);都是以函数语句的方式调用函数。

(3)函数实参：函数作为另一个函数调用的实际参数出现。这种情况是把该函数的返回值作为实参进行传送，因此要求该函数必须是有返回值的。例如：r=max(c, max(a, b));其中 max(a, b)是一次函数调用，它的值作为 max 函数另一次调用的实参。

5.3.2 对被调用函数的声明和函数原型



活动 7 观察主调函数与被调用函数的位置关系。

请读者观察例 5.8，思考以下几个问题：

(1)如果将这个例子中的被调函数 SQ 放在主调函数 main 之后，那么将会出现什么结果？

(2)如果一定要将 SQ 放在 main 之后我们应该怎么写才会出现我们期待的正确结果？

经过函数顺序调换之后，读者会发现程序在编译阶段就出错了，VC++编译器提示出现以下错误：

```
error C2065: 'SQ' : undeclared identifier
error C2373: 'SQ' : redefinition; different typemodifiers
```

仅仅是函数位置的调换便出现了两个错误，我们发现第一个错误提示 SQ 函数未定义，而 SQ 在程序中第一次出现是在 main 函数里面，也就是说 main 函数在调用 SQ 时找不到被调函数的定义。第二个错误提示 SQ 重定义并且为不同类型，我们注意到 SQ 在定义前

更多高清PDF图书 www.gqpdf.com

已经被调用了一次，所以接下去编译器扫描到真正定义 SQ 函数的地方时误以为 SQ 被重定义了，而且类型和原来出现了类型不一致，因为我们调用 SQ 的时候根本就不用指明它的类型，所以当然就会出现类型不一致的错误了。很明显，第二个错误是第一个错误造成的。我们从这个尝试中可以发现，在主调函数中调用某函数之前应对该被调函数进行说明（声明），这与使用变量之前要先进行变量说明是一样的。在主调函数中对被调函数作说明的目的是使编译系统知道被调函数返回值的类型，以便在主调函数中按此种类型对返回值作相应的处理。

其一般形式为：

类型说明符 被调函数名（类型 形参，类型 形参…）；

或为：

类型说明符 被调函数名（类型，类型…）；

括号内给出了形参的类型和形参名，或只给出形参类型。这便于编译系统进行检错，以防止可能出现的错误。

因此如果一定要把例 5.8 的被调函数 SQ 写在主函数后面则必须对 SQ 声明。

【例 5.11】 对被调函数的声明。

【程序】

```
main()
{
  int i=1;
  int SQ(int y);
  while(i<=5)
      printf("%d\n",SQ(i++));
}
SQ(int y)
{
  return((y)*(y));
}
```

【分析】 例 5.11 中的函数说明“int SQ(int y);”也可以写成“int SQ(int);”的形式，即可以省略形参名。

C 语言中规定在以下几种情况时可以省去主调函数中对被调函数的函数说明。

(1)如果被调函数的返回值是整型或字符型时，可以不对被调函数作说明，而直接调用。这时系统将自动对被调函数返回值按整型处理。

(2)当被调函数的函数定义出现在主调函数之前时，在主调函数中也可以不对被调函数再作说明而直接调用。例如例 5.8 中，函数 SQ 的定义放在 main 函数之前，因此可在 main 函数中省去对 SQ 函数的函数说明 int SQ(int y)。

(3)如在所有函数定义之前，在函数外预先说明了各个函数的类型，则在以后的各主调函数中，可不再对被调函数作说明。例如：

```
char str(int a);
float f(float b);
```

高清PDF原创 www.gqpdf.com

```
main()
{
  ......
}
char str(int a)
{
  ......
}
float f(float b)
{
  ......
}
```

其中第一,二行对 str 函数和 f 函数预先作了说明。因此在以后各函数中无须对 str 和 f 函数再作说明就可直接调用。

(4)对库函数的调用不需要再作说明,但必须把该函数的头文件用 include 命令包含在源文件前部。

即时练习

编程实现输入 10 个学生的姓名、三科成绩,自动计算出总分,按总分的从高到低排序后,输出 10 个学生的信息。要求:除 main()外另用三个函数实现。如 input()实现输入,output()实现输出,sort()实现排序。

5.3.3 函数的嵌套调用

读者仔细观察例 5.10 就会发现我们在求三个数的最大值时先通过 main 函数调用 max 函数,然后 max 函数又调用了一次 max 函数作为参数使用,在 C 程序设计中,被调函数中可以再次调用其他函数(包括它自己)从而形成多层的函数调用,我们称这种多层的函数调用为函数的嵌套调用。C 语言中不允许作嵌套的函数定义。因此各函数之间是平行的,不存在上一级函数和下一级函数的问题。但是 C 语言允许在一个函数的定义中出现对另一个函数的调用。这样就出现了函数的嵌套调用。即在被调函数中又调用其他函数。这与其他语言的子程序嵌套的情形是类似的。函数的嵌套调用如图 5.1 所示:

图 5.1 的执行流程为:

①执行 main 函数的开头部分;

②遇调用函数 a 的操作语句,流程转去 a 函数;

③开始执行 a 函数;

④遇调用 b 函数的操作语句,流程转去函数 b;

⑤执行 b 函数,如果再无其他嵌套的函数,则完成 b 函数的全部操作;

⑥返回调用 b 函数处,即返回 a 函数中调用 b 函数的语句处;

⑦继续执行 a 函数中尚未执行的部分,直到 a 函数结束;

更多高清PDF图书 www.gqpdf.com

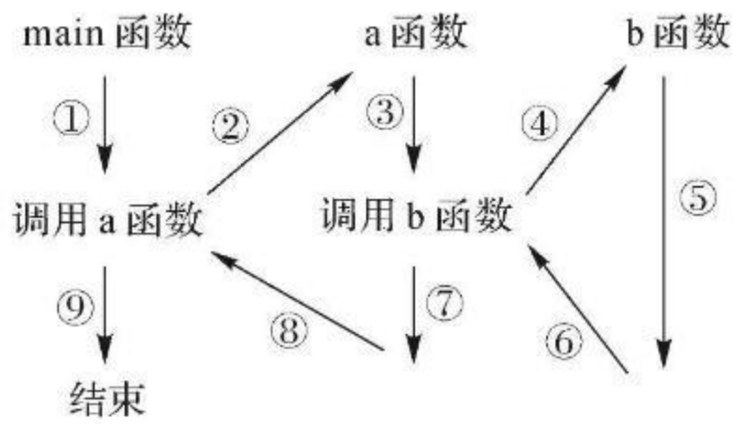


图 5.1　函数的嵌套调用

⑧返回 main 函数中调用 a 函数处；

⑨继续执行 main 函数的剩余部分直到结束。

下面我们通过一道例题进一步加深对函数嵌套的理解。

【例 5.12】 计算 $s=2^2!+3^2!$

【分析】 本题可编写两个函数，一个是用来计算平方值的函数 f1，另一个是用来计算阶乘值的函数 f2。主函数先调 f1 计算出平方值，再在 f1 中以平方值为实参，调用 f2 计算其阶乘值，然后返回 f1，再返回主函数，在循环程序中计算累加和。

【程序】

```
long f1(int p)/*求平方后再求阶乘的函数*/
{
  int k;
  long r;
  long f2(int);/*对 f2 的声明*/
  k=p*p;/*求平方*/
  r= f2(k);/*求阶乘*/
  return r;
}
long f2(int q)/*求阶乘的函数*/
{
  long c=1;
  int i;
  for(i=q;i>=1;i--)/*通过循环求阶乘*/
      c=c* i;
  return c;
}
main()
{
  int i;
  long s=0;
  for(i=2;i<=3;i++)
      s=s+f1(i);
  printf("\ns=%ld\n",s);
}
```

高清PDF原创 www.gqpdf.com

在程序中,函数 f1 和 f2 均为长整型,都在主函数之前定义,故不必再在主函数中对 f1 和 f2 加以说明。在主程序中,执行循环程序依次把 i 值作为实参调用函数 f1 求 i^2 值。在 f1 中又发生对函数 f2 的调用,这时是把 i^2 的值作为实参去调用 f2,在 f2 中完成求 i^2 !的计算。f2 执行完毕后把 C 值(即 i^2 !)返回给 f1,再由 f1 返回主函数实现累加。至此,由函数的嵌套调用实现了题目的要求。由于数值很大,所以需将函数和一些变量的类型都说明为长整型,否则会造成计算错误。

5.3.4 函数的递归调用

活动 8 体会用递归方法求解问题的思路。

请读者仔细阅读如下的问题、列出该问题的数学通式并尝试编写函数解决该问题。

【例 5.13】 有 5 个人坐在一起,问第 5 个人多少岁?他说比第 4 个人大 2 岁。问第 4 个人多少岁?他说比第 3 个人大 2 岁。问第 3 个人多少岁?他说比第 2 个人大 2 岁。问第 2 个人多少岁?他说比第 1 个人大 2 岁。最后问第 1 个人多少岁?他说是 10 岁。请问第 5 个人多大。

【分析】 这是个简单的数学问题,要求第 5 个人的年龄必须先知道第 4 个人的年龄,而第 4 个人的年龄也不知道,要求第 4 个人的年龄必须先知道第 3 个人的年龄,而第 3 个人的年龄又取决于第 2 个人的年龄,第 2 个人的年龄取决于第 1 个人的年龄。而且每一个人的年龄都比其前 1 个人的年龄大 2。按照我们平常的思维方式,应该是先求第 2 个人的年龄,再求第 3 个人的年龄,再求第 4 个人的年龄,再求第 5 个人的年龄,这种方式叫递推法,递推法的特点是从一个已知的事实出发,按一定规律推出下一个事实,再从这个新的已知的事实出发,再向下推出一个新的事实,由此不断推导得到最终解。递推法在解决较小问题的情况下是没问题的,但是当问题规模扩大时,我们有时候很难找到最初的那个子问题(在这里是第 1 个人的年龄)。所以我们不得不做的事情是先将大的问题逐步化解为小的问题,逐步推导,最后将小的问题逐步回归到大的问题,自然便得到大问题的解了,下面我们尝试用这种思路来解决问题,我们先定义一个 age(int n)的函数,这个函数是表示某个人的年龄,参数 n 表示第 n 人,得到如下的推导:

age(5)=age(4)+2
age(4)=age(3)+2
age(3)=age(2)+2
age(2)=age(1)+2
age(1)=10

我们从这几条式子可以得到数学通式:

age(n)=10 (n=1)
age(n)=age(n−1)+2 (n>1)

图 5.2 表示求第 5 个人年龄的过程。

由图 5.2 可知,求解可分成两个阶段:

更多高清PDF图书 www.gqpdf.com

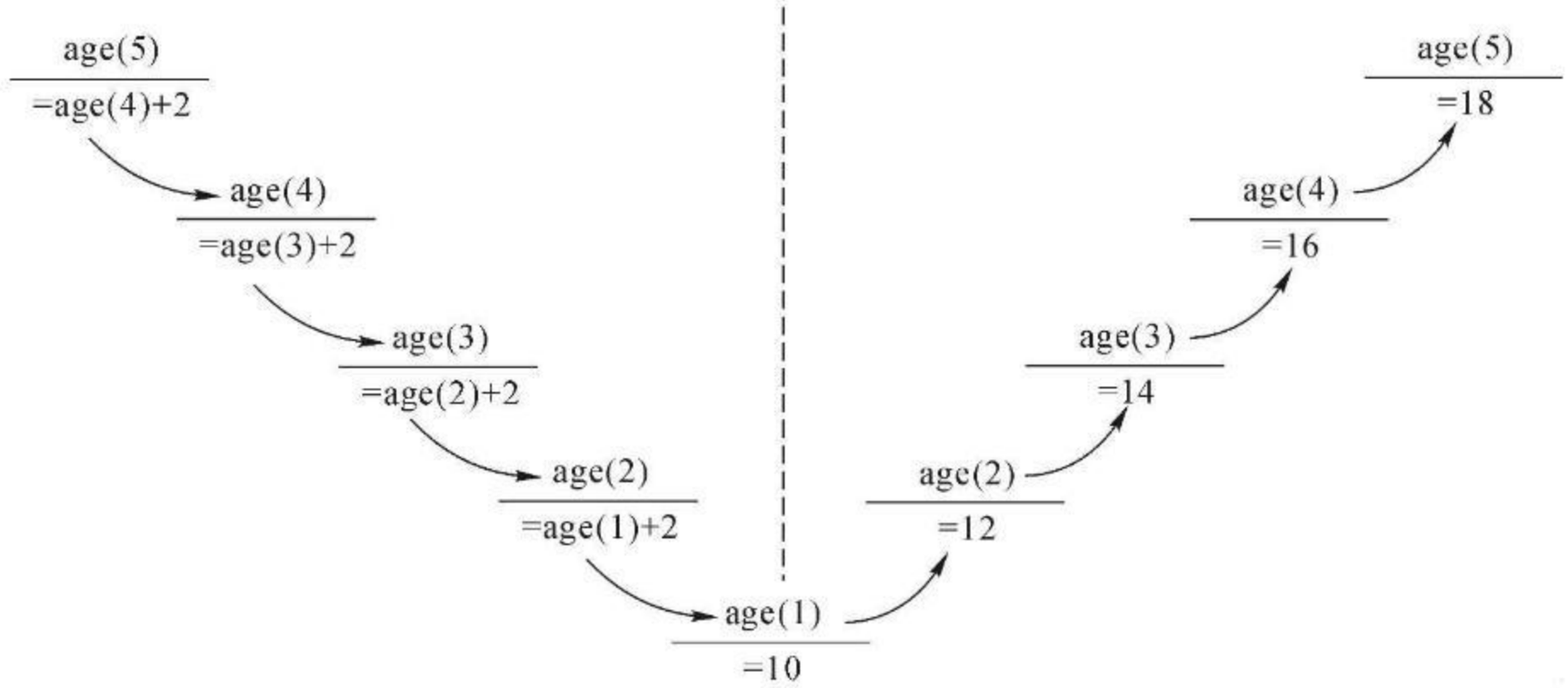


图 5.2 求第 5 个人年龄的过程

第一阶段是“回推”,即将第 n 个人的年龄表示为第(n－1)个人的年龄的函数,而第(n－1)个人的年龄仍然不知道,还要“回推”到第(n－2)个人的年龄……直到第 1 个人年龄。此时 age(1)已知,不必再向前推了。

第二阶段是“递推”,从第 1 个人的已知年龄推算出第 2 个人的年龄(12 岁),从第 2 个人的年龄推算出第 3 个人的年龄(14 岁)……一直推算出第 5 个人的年龄(18 岁)为止。

我们在上述解题的过程中用到的方法实质上是递归的方法。因为“回推”和“递推”是用递归方法求解问题的两个阶段。下面我们给出实现该问题的源代码:

【程序】

```
int age(int n)     /* 求年龄的递归函数 */
{
  int c;           /* c用作存放函数的返回值的变量 */
  if(n == 1)
     c=10;
  else
     c=age(n-1)+2;
  return c;
}
void main()
{
  printf("%d\n", age(5));
}
```

函数调用过程如图 5.3 所示:

下面我们给出递归的相关定义。

一个函数在它的函数体内调用它自身称为递归调用。这种函数称为递归函数。C 语言允许函数的递归调用。在递归调用中,主调函数又是被调函数。执行递归函数将反复调用其自身,每调用一次就进入新的一层。如果要求递归过程不是无限制进行下去,必须具有一

高清PDF原创 www.gqpdf.com

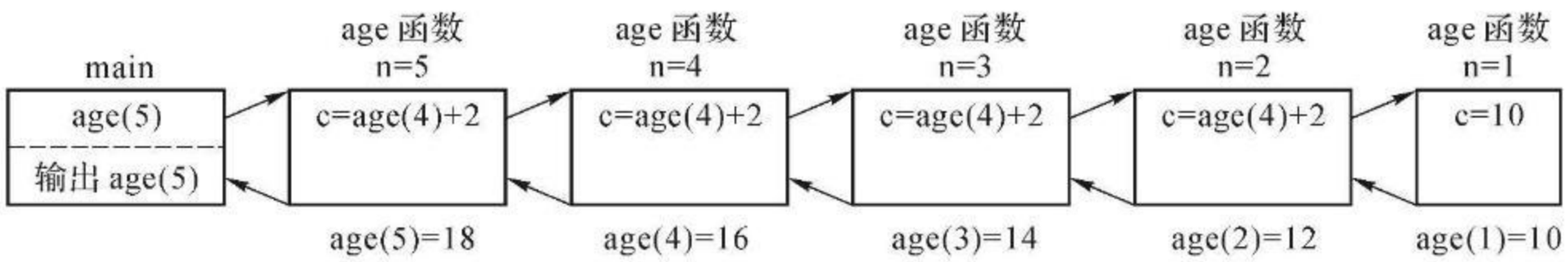


图 5.3 函数调用过程

个结束递归过程的条件。常用的办法是加条件判断,满足某种条件后就不再作递归调用,然后逐层返回。下面我们通过例 5.14 加深对递归的印象。

【例 5.14】 用递归法计算 $n!$

【分析】 可以容易得到该问题的通式是:

$$n!=\begin{cases}1 & (n=0,1)\\ n\times(n-1)! & (n>1)\end{cases}$$

由通式我们可以编写程序如下:

【程序】

```
long ff(int n)
{
  long f;
  if(n<0) printf("n<0,input error");
  else if(n==0||n==1) f=1;
  else f=ff(n-1)*n;
  return(f);
}
main()
{
  int n;
  long y;
  printf("\ninput a inteager number:\n");
  scanf("%d",&n);
  y=ff(n);
  printf("%d!=%ld",n,y);
}
```

高清PDF原创 www.gqpdf.com

程序中给出的函数 ff 是一个递归函数。主函数调用 ff 后即进入函数 ff 执行,如果 n<0,n==0 或 n=1 时都将结束函数的执行,否则就递归调用 ff 函数自身。由于每次递归调用的实参为 n-1,即把 n-1 的值赋予形参 n,最后当 n-1 的值为 1 时再作递归调用,形参 n 的值也为 1,将使递归终止。然后可逐层退回。

下面我们再举例说明该过程。设执行本程序时输入为 5,即求 5!。在主函数中的调用语句即为 y=ff(5),进入 ff 函数后,由于 n=5,不等于 0 或 1,故应执行 f=ff(n-1)*n,即 f=ff(5-1)*5。该语句对 ff 作递归调用即 ff(4)。

进行四次递归调用后,ff 函数形参取得的值变为 1,故不再继续递归调用而开始逐层返

更多高清PDF图书 www.gqpdf.com

回主调函数。ff(1)的函数返回值为1,ff(2)的返回值为1＊2＝2,ff(3)的返回值为2＊3＝6,ff(4)的返回值为6＊4＝24,最后返回值ff(5)为24＊5＝120。

即时练习

用递归方法编写函数pow(float x,int y)求x的y次方。

5.4　变量的作用域与存储类别

5.4.1　局部变量与全局变量

活动9　区分局部变量与全局变量的用法和作用域。

请读者运用函数的知识编写程序实现如下功能：

输入长方体的长宽高l,w,h。求体积及三个面x＊y,x＊z,y＊z的面积。

分析题目我们发现输入的值有三个而输出的值有四个,根据我们所学的知识函数只有一个返回值,所以要实现题目的要求我们必须有四个功能不同的函数来解决这个问题。

【例5.15】 用函数求长方体体积及三个面x＊y,x＊z,y＊z的面积。

【程序】

```
int vs( int a,int b,int c)
{
  int v;
  v=a*b*c;
  return v;
}
int sab( int a,int b,int c)
{
  int s1;
  s1=a*b;
  return s1;
}
int sbc( int a,int b,int c)
{
  int s2;
  s2=b*c;
  return s2;
}
int sac( int a,int b,int c)
{
```

高清PDF原创 www.gqpdf.com

```
    int s3;
    s3=a*c;
    return s3;
}
main()
{
    int v,l,w,h,s1,s2,s3;
    printf("\ninput length,width and height\n");
    scanf("%d%d%d",&l,&w,&h);
    v=vs(l,w,h);
    s1=sab(l,w,h);
    s2=sbc(l,w,h);
    s3=sac(l,w,h);
    printf("\nv=%d,s1=%d,s2=%d,s3=%d\n",v,s1,s2,s3);
}
```

在例 5.15 中,我们使用函数 vs 求长方体的体积,用函数 sab、sbc 和 sac 求三个面的面积。注意在 main 函数中定义的 s1、s2 和 s3 跟函数 sab、sbc 和 sac 里定义的 s1、s2 和 s3 是不同的,在函数 sab、sbc 和 sac 里定义的三个变量我们叫做局部变量,它们只在定义自己的函数里面有效,主函数里定义的变量也具有相同的性质。

我们发现上述例题用了 4 个函数得到 4 个返回值,如果题目比较复杂要求有多个类型不同的返回值是不是也要写很多个函数来实现这个功能呢?我们会发现随着输出的增多函数会变得很多从而影响程序的可读性和可维护性。在下一章我们将会采用数组解决类型相同的多个返回值的问题,但是对于类型不同的多个返回值,数组将会无能为力。我们希望有这么一种变量,在定义之后每个函数都可以使用它,并且可以通过某个函数改变它的值最后在主函数中将它的值输出,C 语言提供了全局变量帮我们解决这个问题。

【例 5.16】 用全局变量求长方体体积及三个面 x * y,x * z,y * z 的面积。

【程序】

```
int s1,s2,s3;
int vs( int a,int b,int c)
{
    int v;
    v=a*b*c;
    s1=a*b;
    s2=b*c;
    s3=a*c;
    return v;
}
main()
{
    int v,l,w,h;
    printf("\ninput length,width and height\n");
```

更多高清PDF图书 www.gqpdf.com

```
    scanf("%d%d%d",&l,&w,&h);
    v=vs(l,w,h);
    printf("\nv=%d,s1=%d,s2=%d,s3=%d\n",v,s1,s2,s3);
}
```

由例5.16可以看出,我们定义了三个全局变量s1,s2和s3并通过函数vs改变了它们的值从而求出三个面的面积,这种解决方法让程序的可读性变强,更容易解决问题。

在接触过局部变量和全局变量之后我们给出它们的定义和使用说明。

1.局部变量的定义

局部变量也称为内部变量。局部变量是在函数内作定义说明的。其作用域仅限于函数内,离开该函数后再使用这种变量是非法的。

例如:

```
int f1(int a)          /* 函数 f1 */
{
  int b,c;
  ......
}
```

a,b,c有效。

```
int f2(int x)          /* 函数 f2 */
{
  int y,z;
  ......
}
```

x,y,z有效。

```
main()
{
  int m,n;
  ......
}
```

m,n有效。

在函数f1内定义了三个变量,a为形参,b、c为一般变量。在f1的范围内a,b,c有效,或者说a,b,c变量的作用域限于f1内。同理,x,y,z的作用域限于f2内。m,n的作用域限于main函数内。

2.局部变量的使用说明

(1)主函数中定义的变量也只能在主函数中使用,不能在其他函数中使用。同时,主函数中也不能使用其他函数中定义的变量。因为主函数也是一个函数,它与其他函数是平行关系。这一点是与其他语言不同的,应予以注意。

(2)形参变量是属于被调函数的局部变量,实参变量是属于主调函数的局部变量。

(3)允许在不同的函数中使用相同的变量名,它们代表不同的对象,分配不同的单元,互不干扰,也不会发生混淆。如在前例中,形参和实参的变量名都为 n,是完全允许的。

(4)在复合语句中也可定义变量,其作用域只在复合语句范围内。

例如:

```
main()
{
  int s,a;
  ……
  {
      int b;
      s=a+b;
      ……          /* b作用域 */
  }
      ……          /* s,a作用域 */
}
```

3.全局变量

全局变量也称为外部变量,它是在函数外部定义的变量。它不属于哪一个函数,它属于一个源程序文件。其作用域是整个源程序。在函数中使用全局变量,一般应作全局变量说明。只有在函数内经过说明的全局变量才能使用。全局变量的说明符为 extern。但在一个函数之前定义的全局变量,在该函数内使用可不再加以说明。

例如:

```
int a,b;          /* 外部变量 */
void f1()         /* 函数 f1 */
{
  ……
}
float x,y;        /* 外部变量 */
int fz()          /* 函数 fz */
{
  ……
}
main()            /* 主函数 */
{
  ……
}
```

从上例可以看出 a、b、x、y 都是在函数外部定义的外部变量,都是全局变量。但 x,y 定义在函数 f1 之后,而在 f1 内又无对 x,y 的说明,所以它们在 f1 内无效。a,b 定义在源程序最前面,因此在 f1,f2 及 main 内不加说明也可使用。

更多高清PDF图书 www.gqpdf.com

如果同一个源文件中,外部变量与局部变量同名,则在局部变量的作用范围内,外部变量被“屏蔽”,即它不起作用。

即时练习

小组讨论全局变量和局部变量的优点和缺点并编程演示为何全局变量的不当使用会存在安全问题。

5.4.2　变量的存储类别

1.动态存储方式与静态动态存储方式

我们前面已经介绍了,从变量的作用域(即从空间)角度来分,可以分为全局变量和局部变量。现在我们从另一个角度,即从变量值存在的作时间(即生存期)角度来分,可以分为静态存储方式和动态存储方式。

静态存储方式:是指在程序运行期间分配固定的存储空间的方式。

动态存储方式:是在程序运行期间根据需要进行动态的分配存储空间的方式。

用户存储空间可以分为三个部分:

(1)程序区;

(2)静态存储区;

(3)动态存储区。

全局变量全部存放在静态存储区,在程序开始执行时给全局变量分配存储区,程序执行完毕就释放。在程序执行过程中它们占据固定的存储单元,而不动态地进行分配和释放。

动态存储区存放以下数据:

(1)函数形式参数;

(2)自动变量(未加 static 声明的局部变量);

(3)函数调用时的现场保护和返回地址。

对以上这些数据,在函数开始调用时分配动态存储空间,函数结束时释放这些空间。

在 C 语言中,每个变量和函数有两个属性:数据类型和数据的存储类别。

2.auto 变量

函数中的局部变量,如不专门声明为 static 存储类别,都是动态地分配存储空间的,数据存储在动态存储区中。函数中的形参和在函数中定义的变量(包括在复合语句中定义的变量),都属此类,在调用该函数时系统会给它们分配存储空间,在函数调用结束时就自动释放这些存储空间。这类局部变量称为自动变量。自动变量用关键字 auto 作存储类别的声明。

例如:

```
int f(int a)            /*定义 f函数,a为参数*/
{
```

```
    auto int b,c=3;  /*定义 b,c自动变量*/
    ……
}
```

a 是形参,b、c 是自动变量,对 c 赋初值 3。执行完 f 函数后,自动释放 a、b、c 所占的存储单元。

关键字 auto 可以省略,auto 不写则默认为“自动存储类别”,属于动态存储方式。

3.用 static 声明局部变量

有时希望函数中的局部变量的值在函数调用结束后不消失而保留原值,这时就应该指定局部变量为“静态局部变量”,用关键字 static 进行声明。

【例 5.17】 考察静态局部变量的值。

【程序】

```
f(int a)
{
  auto b=0;
  static c=3;
  b=b+1;
  c=c+1;
  return(a+b+c);
}
main()
{
  int a=2,i;
  for(i=0;i<3;i++)
     printf("%d",f(a));
}
```

程序运行的结果是 7 8 9,我们发现每一次 c 的值都在原有值上加 1 因为它是静态的变量,而 b 的存储类型由于是 auto 的所以每一次都会重新分配空间并赋值为 0 再加 1。

对静态局部变量的说明:

(1)静态局部变量属于静态存储类别,在静态存储区内分配存储单元。在程序整个运行期间都不释放。而自动变量(即动态局部变量)属于动态存储类别,占动态存储空间,函数调用结束后即释放。

(2)静态局部变量在编译时赋初值,即只赋初值一次;而对自动变量赋初值是在函数调用时进行,每调用一次函数重新给一次初值,相当于执行一次赋值语句。

(3)如果在定义局部变量时不赋初值的话,则对静态局部变量来说,编译时自动赋初值 0(对数值型变量)或空字符(对字符变量)。而对自动变量来说,如果不赋初值则它的值是一个不确定的值。

4.用 extern 声明外部变量

外部变量(即全局变量)是在函数的外部定义的,它的作用域为从变量定义处开始,到本

更多高清PDF图书 www.gqpdf.com

程序文件的末尾。如果外部变量不在文件的开头定义,其有效的作用范围只限于定义处到文件终了。如果在定义点之前的函数想引用该外部变量,则应该在引用之前用关键字extern对该变量作“外部变量”声明。表示该变量是一个已经定义的外部变量。有了此声明,就可以从“声明”处起,合法地使用该外部变量。

【例5.18】 用extern声明外部变量,扩展程序文件中的作用域。

【程序】

```
intmax(intx,inty)
{
  intz;
  z=x>y? x:y;
  return(z);
}
main()
{
  externA,B;
  printf("%d\n",max(A,B));
}
intA=13,B=-8;
```

在例5.18的最后1行定义了外部变量A,B,但由于外部变量定义的位置在函数main之后,因此本来在main函数中不能引用外部变量A,B。现在我们在main函数中用extern对A和B进行“外部变量声明”,就可以从“声明”处起,合法地使用该外部变量A和B。

5.register变量

为了提高效率,C语言允许将局部变量的值放在CPU的寄存器中,这种变量叫“寄存器变量”,用关键字register做声明。

说明:

(1)只有局部自动变量和形式参数可以作为寄存器变量;

(2)一个计算机系统中的寄存器数目有限,不能定义任意多个寄存器变量;

(3)局部静态变量不能定义为寄存器变量。

5.5　本章小结

本章介绍了C语言函数的使用方法,函数就是将实现特定功能的代码封装在一起进行命名,可实现多次调用的子程序。在C语言中可从不同的角度对函数分类。函数的参数和返回值是本章重点讨论的问题,在此基础上我们学习了函数的一般调用、嵌套调用和递归调用。最后一节介绍了变量的作用域和存储类别,这些都是函数的基本知识,请读者融会贯通,并能做到灵活运用。

高清PDF原创 www.gqpdf.com

课后练习

1.编写两个函数,分别求两个整数的最大公约数和最小公倍数,用主函数调用这两个函数并输出结果。

2.写一判断素数的函数并输出判断的结果。

3.写一函数,输入一个二进制数,输出相应的十进制数。

4.输入 10 个学生 5 门课的成绩,要求用函数实现:

(1)求出每个学生的平均分。(2)每门课的平均分。

5.使用递归的方法解决汉诺塔问题:一块板上有三根针,A,B,C。A 针上套有 64 个大小不等的圆盘,大的在下,小的在上。要把这 64 个圆盘从 A 针移动 C 针上,每次只能移动一个圆盘,移动可以借助 B 针进行。但在任何时候,任何针上的圆盘都必须保持大盘在下,小盘在上。求移动的步骤。

高清PDF原创 www.gqpdf.com
更多高清PDF图书 www.gqpdf.com

高清PDF原创 www.gqpdf.com

第 6 章

数　组

【本章主要内容】

本章主要介绍一维数组的定义和使用方法、二维数组的定义和使用方法、字符数组的使用方法和常用函数、数组作为函数参数等数组的知识，读者学完本章后可对数组有详细的了解并能使用数组简化程序的设计，提高编程的效率。

【本章要求】

(1)了解数组作为函数参数使用的基本方法。
(2)理解字符数组常用函数的使用方法。
(3)掌握一维数组、二维数组、字符数组的定义和使用方法。

6.1　一维数组

活动 1　讨论常用变量类型的局限性。

请读者设计程序实现如下功能：

输入 10 个整数并按逆序输出这 10 个数。

按照我们所学过的知识，读者可能觉得这个程序的解决方法非常简单，先定义 10 个整型变量然后逐个读入值并根据读入的逆序一一输出读入的值。如果按照这种方法，我们输入要写 10 条语句，而输出也要相应写 10 条语句。而在实际应用中，可能会有成千上万个数据要求输入和输出。显然，用我们所学过的数据类型进行逐个的输入和输出是不现实的。我们发现这里 10 个数都是整型，因此可以用 C 语言提供的另一种数据类型来实现此题的功能。

在程序设计中，为了处理方便，把具有相同类型的若干变量按有序的形式组织起来。这

些按序排列的同类数据元素的集合称为数组。在 C 语言中,整型、字符型、实型都是属于基本类型的数据,数组属于构造数据类型。一个数组可以分解为多个数组元素,这些数组元素可以是基本数据类型或是构造类型。因此按数组元素的类型不同,数组又可分为数值数组、字符数组、指针数组、结构数组等各种类别。

6.1.1 一维数组的定义

一维数组即是维度为一的数组,相当于将类型相同的若干个变量排成一列。在 C 语言中使用数组前必须先进行定义。

一维数组的定义方式为:

类型说明符 数组名[常量表达式];

其中:

类型说明符是任一种基本数据类型或构造数据类型。

数组名是用户定义的数组标识符。

方括号中的常量表达式表示数据元素的个数,也称为数组的长度。

例如:

```
int a[10];              /* 说明整型数组 a,有 10 个元素。 */
float b[10],c[20];      /* 说明实型数组 b,有 10 个元素,实型数组 c,有 20 个元素。 */
char ch[20];            /* 说明字符数组 ch,有 20 个元素。 */
```

对于数组类型说明应注意以下几点:

(1)数组的类型实际上是指数组元素的取值类型。对于同一个数组,其所有元素的数据类型都是相同的。

(2)数组名的书写规则应符合标识符的书写规定。

(3)数组名不能与其他变量名相同。

例如:

```
main()
{
   int a;
   float a[10];
   ......
}
```

是错误的。

(4)方括号中常量表达式表示数组元素的个数,如 a[5]表示数组 a 有 5 个元素。但是其下标从 0 开始计算。因此 5 个元素分别为 a[0],a[1],a[2],a[3],a[4]。

(5)不能在方括号中用变量来表示元素的个数,但是可以是符号常数或常量表达式。

例如:

```
#define FD 5
main()
```

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

```
{
   int a[3+2],b[7+FD];
   ......
}
```

是合法的。

但是下述说明方式是错误的。

```
main()
{
   int n=5;
   int a[n];
   ......
}
```

(6)允许在同一个类型说明中,说明多个数组和多个变量。

例如:

```
int a,b,c,d,k1[10],k2[20];
```

6.1.2　一维数组的引用



活动2　利用一维数组解决活动1的问题。

我们已经了解了一维数组的定义方式,因此可以定义一个具有10个元素的数组来解决问题。定义完数组后我们必须使用数组中的元素,数组元素是组成数组的基本单元。数组元素也是一种变量,其标识方法为数组名后跟一个下标。下标表示元素在数组中的顺序号。

数组元素的一般形式为:

数组名[下标]

【例6.1】 顺序输入10个整数并将这10个整数逆序输出。

【程序】

```
main()
{
   int i,a[10];
   for(i=0;i<=9;i++)
      scanf("%d",&a[i]);
   for(i=9;i>=0;i--)
      printf("%d ",a[i]);
}
```

高清PDF原创 www.gqpdf.com

【分析】 在例6.1中我们先定义了一个循环变量i和一个具有10个元素的数组a,通过第一个for循环输入10个整数,然后通过第二个for循环逆序输出这10个整数。我们发现在输入和输出时我们使用某个数组元素都是用a[i]来表示数组的第i+1个元素(因为数

组元素的下标从 0 开始),其中下标只能为整型常量或整型表达式。如为小数时,编译将自动取整。

例如:

```
a[5]
a[i+j]
a[i++]
```

都是合法的数组元素。

数组元素通常也称为下标变量。必须先定义数组,才能使用下标变量。数组元素可以用于表达式中,也可以被赋值。例如:

```
a[0]=a[5]+a[7]-a[2*3]
```

在 C 语言中只能逐个地使用下标变量,而不能一次引用整个数组,因此我们在例 6.1 中使用循环逐个为数组元素赋值并逐个逆序输出。

即时练习

编写程序输入十个数字然后打印出高于平均分的数字。

6.1.3 一维数组的初始化

活动 3 寻找数组元素赋初值的另一种方式。

在活动 2 中我们给数组元素赋初值的方式是逐个赋值,那么是不是为一维数组元素赋初值一定要逐个来进行呢?有没有一种一次性赋初值的方法呢?答案是肯定的。

我们可以通过例 6.2 来体会这种一次性赋值(或数组的初始化)的方法。

【例 6.2】 一维数组的初始化。

【程序】

```
main()
{
   int i;
   int a[10]={0,1,2,3,4,5,6,7,8,9};
   for(i=9;i>=0;i--)
       printf("%d",a[i]);
}
```

【分析】 我们发现一维数组的赋值可以在定义的时候同时进行,赋值的时候用{ }把所有元素的值括起来,每个元素之间用逗号分隔。下面我们给出一维数组初始化的定义和使用方法。

数组初始化赋值是指在数组定义时给数组元素赋予初值。数组初始化是在编译阶段进行的。这样将减少运行时间,提高效率。

更多高清PDF图书 www.gqpdf.com

初始化赋值的一般形式为：

类型说明符 数组名[常量表达式]={值,值……值};

其中在{ }中的各数据值即为各元素的初值,各值之间用逗号间隔。

例如：

```
int a[10]={0,1,2,3,4,5,6,7,8,9};
```

相当于

```
a[0]=0;a[1]=1...a[9]=9;
```

C 语言对数组的初始化赋值还有以下几点规定：

(1)可以只给部分元素赋初值。

当{ }中值的个数少于元素个数时,只给前面部分元素赋值。

例如：

```
int a[10]={0,1,2,3,4};
```

表示只给 a[0]～a[4]5 个元素赋值,而后 5 个元素自动赋 0 值。

(2)只能给元素逐个赋值,不能给数组整体赋值。

例如给十个元素全部赋 1 值,只能写为：

```
int a[10]={1,1,1,1,1,1,1,1,1,1};
```

而不能写为：

```
int a[10]=1;
```

(3)如给全部元素赋值,则在数组说明中，可以不给出数组元素的个数。

例如：

```
int a[5]={1,2,3,4,5};
```

可写为：

```
int a[]={1,2,3,4,5};
```

6.1.4　一维数组的使用实例



活动 4　输入 10 个整数并将它们按从小到大排列存放在数组中。

本活动的本意其实是让我们将 10 个整数进行排序。在前面的章节中,我们几乎没对排序做过详细的讨论,其实排序在实际应用中非常广泛,目前世界上常用的排序算法就多达二三十种,我们的后续课程“数据结构”将详细讨论排序算法,这里我们给读者介绍一种最基本的排序算法——冒泡排序法。

冒泡排序法的基本思路是:比较相邻两个数,将小的调到前头(小数上浮,大数下沉)。

假设待排序序列里有 6 个数 9—8—5—4—2—0。则第一次将 8 和 9 对调,第二次将第 2 和第 3 个数(9 和 5)对调……如此共进行 5 次,得到 8—5—4—2—0—9 的顺序,这是第一

高清PDF原创 www.gqpdf.com

趟排序。经第一趟排序后最大的数 9 已“沉底”,成为最下面一个数,其他小的数“上升”,如图 6.1 所示。然后进行第二趟排序,对余下的前面 5 个数按上法进行比较,经过 4 次比较,得到次大的数 8,如图 6.2 所示。对 6 个数要比较 5 趟,才能使 6 个数按大小顺序排列。在第一趟中两个数之间的比较要进行 5 次,在第二趟中比较 4 次……第 5 趟中比较 1 次。如果有 n 个数,则要进行 n－1 趟排序。在第 1 趟排序中要进行 n－1 次两两比较,在第 j 趟排序中要进行 n－j 次两两比较。

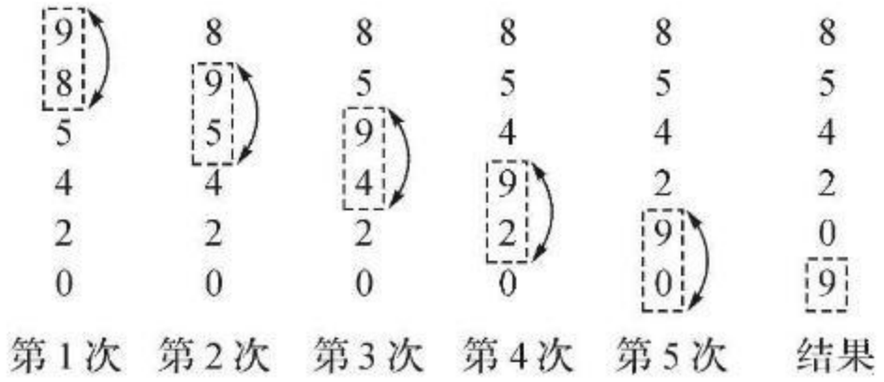


图 6.1 冒泡法第一趟排序

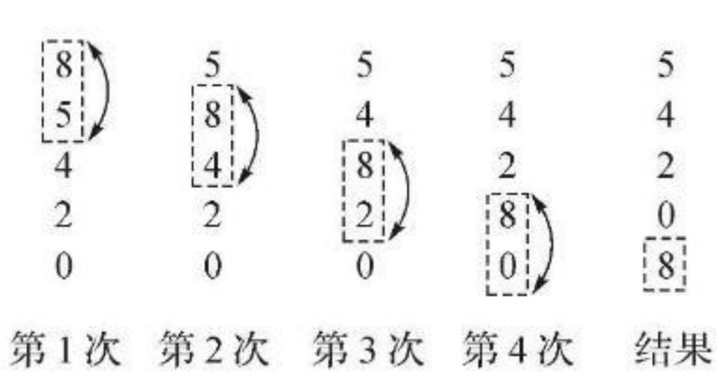


图 6.2 冒泡法第二趟排序

对冒泡算法进行分析后我们可以得到以下程序:

【例 6.3】 冒泡排序

【程序】

```
void Bubblesort()
{
   int a[10];
   int i, j, t;

printf("input 10 numbers:\n");
for(i=0; i<10; i++)
    scanf("%d",&a[i]);//输入

printf("\n");

for(j=0; j<9; j++)
    for(i=0; i<10-j; i++)
        if(a[i]>a[i+1])//大数沉底
        {
            t=a[i];
            a[i]=a[i+1];
            a[i+1]=t;
        }

printf("the sorted numbers:\n");

for(i=0; i<10; i++)
    printf("%d",a[i]);//排序后输出
```

更多高清PDF图书 www.gqpdf.com

```
    printf("\n");
}
voidmain()
{
    Bubblesort();
}
```

本程序中间的嵌套循环正是冒泡排序的经典写法,请读者仔细分析,慢慢品味。

即时练习

输入一串由数字和字母组成的字符串(要求至少要有一个数字),将所有数字从大到小存放到数组中并将排好序后的数字打印出来。

6.2 二维数组

6.2.1 二维数组的定义

活动5 一个学习小组有5个人,每个人有三门课的考试成绩,如表6.1所示。求全组每科的平均成绩和各科总平均成绩。

表6.1 某学习小组考试成绩情况表

	赵 一	钱 二	孙 三	李 四	王 五
C语言	80	61	59	85	76
数据结构	75	65	63	87	77
操作系统	92	71	70	90	85

高清PDF原创 www.gaopdf.com

我们根据学过的一维数组的知识可以定义三个一维数组,分别对应三门课,每个数组5个元素,这样便能实现活动5要求的功能,但是细心的读者会发现这张表其实是张二维表,即表是由行和列组成的,很多实际的问题都是用二维表来表示的。C语言其实已经为我们提供了二维的数据表示方法——二维数组。

二维数组定义的一般形式是:

类型说明符 数组名[常量表达式1][常量表达式2]

其中常量表达式1表示第一维下标的长度,常量表达式2表示第二维下标的长度。

例如:

```
int a[3][4];
```

说明了一个三行四列的数组,数组名为a,其下标变量的类型为整型。该数组的下标变

量共有 3×4 个,即:

```
a[0][0],a[0][1],a[0][2],a[0][3]
a[1][0],a[1][1],a[1][2],a[1][3]
a[2][0],a[2][1],a[2][2],a[2][3]
```

二维数组在概念上是二维的,即其下标在两个方向上变化,下标变量在数组中的位置也处于一个平面之中,而不是像一维数组只是一个向量。但是,实际的硬件存储器却是连续编址的,也就是说存储器单元是按一维线性排列的。如何在一维存储器中存放二维数组,可有两种方式:一种是按行排列,即放完一行之后顺次放入第二行。另一种是按列排列,即放完一列之后再顺次放入第二列。在 C 语言中,二维数组是按行排列的。即:先存放 a[0]行,再存放 a[1]行,最后存放 a[2]行。每行中的四个元素也是依次存放。

在实际问题中有很多量是二维的或多维的,因此 C 语言允许构造多维数组。多维数组元素有多个下标,以标识它在数组中的位置,所以也称为多下标变量。本小节只介绍二维数组,多维数组可由二维数组类推而得到。

6.2.2 二维数组的引用



活动 6 利用二维数组解决活动 5 的问题。

我们可以设一个二维数组 a[5][3]存放五个人三门课的成绩,当然也可以设一个二维数组 b[3][5]存放三门课五个人的成绩,这里我们采用第一种方式。再设一个一维数组 v[3]存放所求得各分科平均成绩,设变量 average 为全组各科总平均成绩。接下来就应该考虑数组元素的引用问题了,二维数组元素的引用和一维数组是相似的,二维数组的元素也称为双下标变量,其表示的形式为:

数组名[下标][下标]

其中下标应为整型常量或整型表达式。

例如:

```
a[3][4]
```

表示 a 数组第四行五列的元素。

下标变量和数组说明在形式中有些相似,但这两者具有完全不同的含义。数组说明的方括号中给出的是某一维的长度,即可取下标的最大值;而数组元素中的下标是该元素在数组中的位置标识。前者只能是常量,后者可以是常量,变量或表达式。

【例 6.4】 使用二维数组求学习小组每科的平均成绩和各科总平均成绩。

【程序】

```
void CalGrade()
{
   int i,j,s=0,average,v[3],a[5][3];
   printf("input score\n");
   for(i=0;i<3;i++)
```

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

```
{
    for(j=0;j<5;j++)
    {
        scanf("%d",&a[j][i]);
        s=s+a[j][i];
    }
    v[i]=s/5;
    s=0;
  }
  average=(v[0]+v[1]+v[2])/3;
  printf("C language:%d\nDataStructure:%d\nOperatingSystem:%d\n",v[0],v[1],v[2]);
  printf("total:%d\n", average);
}
main()
{
  CalGrade();
}
```

【分析】 程序中首先用了一个双重循环。在内循环中依次读入某一门课程的各个学生的成绩,并把这些成绩累加起来,退出内循环后再把该累加成绩除以5送入v[i]之中,这就是该门课程的平均成绩。外循环共循环三次,分别求出三门课各自的平均成绩并存放在v数组之中。退出外循环之后,把v[0],v[1],v[2]相加除以3即得到各科总平均成绩。最后按题意输出各个成绩。

6.2.3　二维数组的初始化

1.二维数组初始化方法

二维数组初始化也是在类型说明时给各下标变量赋以初值。二维数组可按行分段赋值,也可按行连续赋值。

例如对数组a[5][3]:

(1)按行分段赋值可写为:

```
int a[5][3]={ {80,75,92},{61,65,71},{59,63,70},{85,87,90},{76,77,85} };
```

(2)按行连续赋值可写为:

```
int a[5][3]={ 80,75,92,61,65,71,59,63,70,85,87,90,76,77,85};
```

这两种赋初值的结果是完全相同的。

2.二维数组初始化说明

(1)可以只对部分元素赋初值,未赋初值的元素自动取0值。

例如:

```
int a[3][3]={{1},{2},{3}};
```

是对每一行的第一列元素赋值,未赋值的元素取 0 值。赋值后各元素的值为:

```
1 0 0
2 0 0
3 0 0
```

而 `int a[3][3]={{0,1},{0,0,2},{3}};`

赋值后的元素值为:

```
0 1 0
0 0 2
3 0 0
```

(2)如对全部元素赋初值,则第一维的长度可以不给出。

例如:

```
int a[3][3]={1,2,3,4,5,6,7,8,9};
```

可以写为:

```
int a[][3]={1,2,3,4,5,6,7,8,9};
```

(3)数组是一种构造类型的数据。二维数组可以看作是由一维数组的嵌套而构成的。设一维数组的每个元素又都是一个数组,就组成了二维数组。当然,前提是各元素类型必须相同。根据这样的分析,一个二维数组也可以分解为多个一维数组。C 语言允许这种分解。

如二维数组 a[3][4],可分解为三个一维数组,其数组名分别为:

```
a[0]
a[1]
a[2]
```

对这三个一维数组不需另作说明即可使用。这三个一维数组都有 4 个元素,例如:一维数组 a[0]的元素为 a[0][0],a[0][1],a[0][2],a[0][3]。

必须强调的是:a[0],a[1],a[2]不能当做下标变量使用,它们是数组名,不是一个单纯的下标变量。

6.2.4 二维数组的应用



活动 7 设有一个 2 行 3 列的矩阵,试编写程序将这个矩阵进行转置并考虑有没有能实现 $n\times m$ 矩阵转置的通用算法?

根据活动 7 的要求,要将此矩阵进行转置(即行元素变成列元素,列元素变成行元素),则转置后的矩阵应该是 3 行 2 列的,所以我们除了定义一个 2 行 3 列的二维数组存放原矩阵之外还必须定义另一个 3 行 2 列的二维数组存放转置后的矩阵。例如:

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

$$a=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}\quad b=\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}$$

a矩阵转置后便成了b矩阵。由此我们可以编写程序实现转置功能。

【例6.5】 将一个二维数组行和列元素互换,存到另一个二维数组中(矩阵的转置)。

【程序】

```
voidmain()
{
  int a[2][3]={{1, 2, 3}, {4, 5, 6}};
  int b[3][2], i, j;

  printf("array a:\n");
  for(i=0; i<=1; i++)
  {
      for(j=0; j<=2; j++)
      {
          printf("%5d", a[i][j]);
          b[j][i]=a[i][j];
      }
      printf("\n");
  }

  printf("array b:\n");
  for(i=0; i<=2; i++)
  {
      for(j=0; j<=1; j++)
          printf("%5d", b[i][j]);
      printf("\n");
  }

  printf("\n");
}
```

【分析】 算法实现之后我们考虑活动7的最后一个问题:有没有能实现n×m矩阵转置的通用算法?读者可能觉得只需把算法里面的a[2][3]改成a[n][m]而b[3][2]改成b[m][n]就行了。但是在定义二维数组时是不能拿变量做行的元素个数或列的元素个数的,在实际的程序设计中通常的做法是定义一个足够大的数组比如a[1000][1000]然后再输入n和m进行转置。当然也可以用动态数组来实现,我们在学习了指针之后就可以使用动态数组。但是动态数组不利于程序的稳定性,所以有时程序设计是以空间来换取时间或者稳定性的,这在软件工程中是必须考虑的问题。

即时练习

输入一个 3×4 矩阵到二维数组 a,一个 4×2 矩阵到二维数组 b,将 a×b 的结果存储到二维数组 c。

高清PDF原创 www.gqpdf.com

6.3 字符数组

6.3.1 字符数组的定义

用来存放字符变量的数组称为字符数组。字符数组的定义形式与前面介绍的数值数组相同,只是将数组的类型换为字符型,即:

char 数组名[常量表达式];

例如:

```
char c[10];
```

由于字符型和整型通用,即它们在机器里都是用 ASCII 码表示,所以字符数组 char c[10]也可以定义为 int c[10]但这时每个数组元素占 2 个字节的内存单元。

字符数组也可以是二维或多维数组。

例如:

```
char c[5][10];
```

即为二维字符数组。

6.3.2 字符数组的初始化和引用

活动 8 请读者编写程序输出如下所示的图形。

```
    *
   * *
  *   *
 *     *
  *   *
   * *
    *
```

此活动要求我们输出的图形由字符“*”和空格组成,空格也是字符的一种,因此我们使用的基本数据类型应该是字符型。观察图形我们发现这个图形由 5 行 5 列构成,第 1 行由左边两个空格,中间一个 * 号,右边两个空格组成,第 2 行依次由 1 个空格,1 个 * 号,1 个空格,1 个 * 号,1 个空格组成,依次类推……该图形是二维的,所以我们可以用二维的字符数组输出该图形,编程如下:

更多高清PDF图书 www.gqpdf.com

高清PDF原创 www.gqpdf.com

【例 6.6】 字符数组的初始化和引用。

【程序】

```
voidmain()
{
  chardiamond[][5]={{'','','*'},{'','*','','*'},{'*','','','','*'},
{'','*','','*'},{'','','*'}};
  inti, j;
  for(i=0; i< 5; i++)
  {
      for(j=0; j< 5; j++)
          printf("%c",diamond[i][j]);
      printf("\n");
  }
}
```

【分析】 由程序我们可以看出字符数组允许在定义时作初始化赋值。

例如：

```
charc[10]={'c','','p','r','o','g','r','a','m'};
```

赋值后各元素的值为：

c[0]的值为'c'
c[1]的值为''
c[2]的值为'p'
c[3]的值为'r'
c[4]的值为'o'
c[5]的值为'g'
c[6]的值为'r'
c[7]的值为'a'
c[8]的值为'm'

其中c[9]未赋值，由系统自动赋予0值。

当对全体元素赋初值时也可以省去长度说明。

例如：

```
charc[]={'c','','p','r','o','g','r','a','m'};
```

这时C数组的长度自动定为9。

对于二维字符数组，在初始化时可以省略第1维的长度说明，如例6.6所示。

字符数组的引用也与一般数组类似。

6.3.3 字符串和字符串结束标志

活动 9 讨论:C 语言系统如何得到数组的长度?

我们在前面介绍了数组的初始化和使用方式,比如要存放 10 个数据则必须定义一个长度为 10 的数组。在实际应用中,我们往往不知道具体需要存储的数组元素有多少个,因此便预先定义较大的空间来存放数组元素(见例 6.5 的分析),这时要知道数组元素的个数显然便不能依照数组的长度了,那么系统有没有给我们提供解决方法呢? 我们用字符数组来说明。

在 C 语言中没有专门的字符串变量,通常用一个字符数组来存放一个字符串。为了测定字符串的实际长度,C 语言规定了一个“字符串结束标志”,即字符 '\0'。当把一个字符串存入一个数组时,也把结束符 '\0' 存入数组,并以此作为该字符串是否结束的标志。如果有一个字符串,其中第 10 个字符为 '\0',则此字符串的有效字符为 9 个(在遇到字符 '\0' 时,表示字符串结束,由它前面的字符组成字符串)。例如 " C program" 共有 9 个字符,但在内存中占 10 个字节,最后一个字节 '\0' 是由系统自动加上的。字符串作为一维数组存放在内存中。因此有了 '\0' 标志后,就不必再用字符数组的长度来判断字符串的长度了。字符串结束标志对其他类型的数组同样适用。

C 语言允许用字符串的方式对数组作初始化赋值。

例如:

```
char c[]={'c',' ','p','r','o','g','r','a','m'};
```

可写为:

```
char c[]={"C program"};
```

或去掉{}写为:

```
char c[]="C program";
```

用字符串方式赋值比用字符逐个赋值要多占一个字节,用于存放字符串结束标志'\0'。上面的数组 c 在内存中的实际存放情况为:

C		p	r	o	g	r	a	m	\0

'\0' 是由 C 编译系统自动加上的。由于采用了 '\0' 标志,所以在用字符串赋初值时一般无须指定数组的长度,而由系统自行处理。

6.3.4 字符数组的输入输出

在采用字符串方式后,字符数组的输入输出将变得简单方便。

除了上述用字符串赋初值的办法外,还可用 printf 函数和 scanf 函数一次性输出输入一个字符数组中的字符串,而不必使用循环语句逐个地输入输出每个字符。

高清PDF原创 www.gqpdf.com
更多高清PDF图书 www.gqpdf.com

高清PDF原创 www.gqpdf.com

【例 6.7】 字符数组的输入与输出。

【程序】

```
main()
{
   char st[15];
   printf("input string:\n");
   scanf("%s",st);
   printf("%s\n",st);
}
```

【分析】 在例 6.7 的 printf 函数中,使用的格式字符串为“%s”,表示输出的是一个字符串,在输出表列中给出数组名则可。不能写为:

printf("%s",st[]);

例 6.7 中由于定义数组长度为 15,因此输入的字符串长度必须小于 15,以留出一个字节用于存放字符串结束标志 '\0'。应该说明的是,对一个字符数组,如果不作初始化赋值,则必须说明数组长度。还应该特别注意的是,当用 scanf 函数输入字符串时,字符串中不能含有空格,否则将以空格作为串的结束符。

例如当输入的字符串中含有空格时,运行情况为:

```
input string:
this is a book
```

输出为:

```
this
```

从输出结果可以看出空格以后的字符都未能输出。为了避免这种情况,可多设几个字符数组分段存放含空格的串。

程序可改写如下:

【例 6.8】 输入与输出多个字符串。

【程序】

```
main()
{
   char st1[6],st2[6],st3[6],st4[6];
   printf("input string:\n");
   scanf("%s%s%s%s",st1,st2,st3,st4);
   printf("%s%s%s%s\n",st1,st2,st3,st4);
}
```

【分析】 例 6.8 分别设了四个数组,输入的一行字符的空格分段分别装入四个数组。然后分别输出这四个数组中的字符串。

在前面介绍过,scanf 的各输入项必须以地址方式出现,如 &a,&b 等。但在前例中却是以数组名的方式出现的,这是为什么呢?

这是由于在 C 语言中规定,数组名就代表了该数组的首地址。整个数组是以首地址开

头的一块连续的内存单元。

如有字符数组 char c[10],在内存可表示如下：

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]	C[9]

设数组 c 的首地址为 2000,也就是说 c[0]单元地址为 2000。则数组名 c 就代表这个首地址。因此在 c 前面不能再加地址运算符 &。如写作 scanf("%s",&c)则是错误的。在执行函数 printf("%s",c) 时,按数组名 c 找到首地址,然后逐个输出数组中各个字符直到遇到字符串终止标志 '\0' 为止。

即时练习

使用字符数组动态画出一边长可调的正方形(正方形的边长以字符为单位)。边长是动态输入的且最小为 3,如输入 3 表示正方形的边长为 3 个字符长,即 * * * 。

6.3.5 字符串处理函数

活动 10 讨论:在实际应用中字符串有哪些常用的操作?

字符串在实际的应用中十分广泛,比如设计一个软件,在软件开始运行时我们可以让用户输入用户名和密码,这里面我们必须将文本框的用户名进行复制,在复制时需要去掉前后的空格,然后在数据库里面查找该用户,这个过程又用到了字符串的匹配(或者说比较),最后还要检查密码是否正确才能让用户登录。

针对实际应用中如此频繁的字符串操作,C 语言提供了丰富的字符串处理函数,大致可分为字符串的输入、输出、合并、修改、比较、转换、复制、搜索几类。使用这些函数可大大减轻编程的负担。用于输入输出的字符串函数,在使用前应包含头文件"stdio.h",使用其他字符串函数则应包含头文件"string.h"。

下面介绍几个最常用的字符串函数。

1.字符串输出函数 puts

格式:**puts(字符数组名)**

功能:把字符数组中的字符串输出到显示器,即在屏幕上显示该字符串。

2.字符串输入函数 gets

格式:**gets(字符数组名)**

功能:从标准输入设备键盘上输入一个字符串。本函数得到一个函数值,即为该字符数组的首地址。

【例 6.9】 字符串的输入输出函数。

【程序】

```
main()
```

更多高清PDF图书 www.gqpdf.com

```
{
  char st[15];
  char c[]="I like\nC programming!";
  puts(c);
  printf("input string:\n");
  gets(st);
  puts(st);
}
```

【分析】 从例6.9中可以看出puts函数中可以使用转义字符,因此输出结果成为两行。puts函数完全可以由printf函数取代。当需要按一定格式输出时,通常使用printf函数。当输入的字符串中含有空格时,输出仍为全部字符串。说明gets函数并不以空格作为字符串输入结束的标志,而只以回车作为输入结束。这是与scanf函数不同的。

3.字符串连接函数 strcat

格式:**strcat(字符数组名1,字符数组名2)**

功能:把字符数组2中的字符串连接到字符数组1中字符串的后面,并删去字符串1后的串标志"\0"。本函数返回值是字符数组1的首地址。

【例6.10】 字符串的连接函数strcat。

【程序】

```
main()
{
  char st1[30]="My name is ";
  char st2[10];
  printf("input your name:\n");
  gets(st2);
  strcat(st1,st2);
  puts(st1);
}
```

【分析】 例6.10把初始化赋值的字符数组与动态赋值的字符串连接起来。要注意的是,字符数组1应定义足够的长度,否则不能全部装入被连接的字符串。

4.字符串拷贝函数 strcpy

格式:**strcpy(字符数组名1,字符数组名2)**

功能:把字符数组2中的字符串拷贝到字符数组1中。串结束标志"\0"也一同拷贝。字符数名2,也可以是一个字符串常量。这时相当于把一个字符串赋予一个字符数组。本函数要求字符数组1应有足够的长度,否则不能全部装入所拷贝的字符串。

【例6.11】 字符串拷贝函数strcpy

【程序】

```
main()
```

```
{
  char st1[15],st2[]="C Language";
  strcpy(st1,st2);
  puts(st1);printf("\n");
}
```

5.字符串比较函数 strcmp

格式:**strcmp**(**字符数组名 1,字符数组名 2**)

功能:按照 ASCII 码顺序比较两个数组中的字符串,并由函数返回值返回比较结果。

字符串 1=字符串 2,返回值=0

字符串 2 > 字符串 2,返回值 > 0

字符串 1 < 字符串 2,返回值 < 0

本函数也可用于比较两个字符串常量,或比较数组和字符串常量。

【例 6.12】 字符串比较函数 strcmp。

【程序】

```
main()
{
  int k;
  static char st1[15],st2[]="C Language";
  printf("input a string:\n");
  gets(st1);
  k=strcmp(st1,st2);
  if(k==0) printf("st1=st2\n");
  if(k>0) printf("st1>st2\n");
  if(k<0) printf("st1<st2\n");
}
```

【分析】 例 6.12 把输入的字符串和数组 st2 中的串作 ASCII 码比较,比较结果返回到 k 中,根据 k 值再输出结果提示串。

6.测字符串长度函数 strlen

格式:**strlen**(**字符数组名**)

功能:测字符串的实际长度(不含字符串结束标志'\0')并作为函数返回值。

【例 6.13】 测字符串长度函数 strlen。

【程序】

```
main()
{
  int k;
  static char st[]="C language";
  k=strlen(st);
```

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

```
    printf("The lenth of the string is%d\n",k);
}
```

高清PDF原创 www.gqpdf.com

即时练习

输入一字符串到s,将其中的'*'字符移到字符串前面,
如输入:abcd**ef***g**,
则移动后为:*******abcdefg。

6.4 数组作函数参数

6.4.1 数组元素做函数参数



活动11 请读者用函数实现:判别一个整数数组中各元素的值,若大于0则输出该值,若小于等于0则输出0值。

活动中要求实现的功能很简单,问题是必须用函数实现。我们知道如果用函数来实现的话,直观的思维是在函数里面定义一个数组并将数组元素的值判断最后按要求打印出来。但是在函数内定义的数组作用范围只限于函数的内部,如果我们在调用函数以后还要继续使用该数组那应该如何做呢?在实际的应用中,往往是将数组元素作为函数的参数传给函数进行处理的,这样调用完函数之后我们还可以继续使用该数组。

【例6.14】 数组元素做函数参数

【程序】

```
void nzp(int v)
{
  if(v>0)
      printf("%d",v);
  else
      printf("%d",0);
}
main()
{
  int a[5],i;
  printf("input 5 numbers\n");
  for(i=0;i<5;i++)
      scanf("%d",&a[i]);
  for(i=0;i<5;i++)
      nzp(a[i]);
}
```

【分析】 例 6.14 首先定义一个无返回值函数 nzp,并说明其形参 v 为整型变量。在函数体中根据 v 值输出相应的结果。在 main 函数中用一个 for 语句输入数组各元素,用另一个 for 语句调用 nzp 函数,把 a[i]的值传送给形参 v,供 nzp 函数使用,可以看出调用 nzp 的次数和数组元素的个数相同。因此,我们可以做如下总结。

数组元素就是下标变量,它与普通变量并无区别。因此它作为函数实参使用与普通变量是完全相同的,在发生函数调用时,把作为实参的数组元素的值传送给形参,实现单向的值传送。

6.4.2 数组名做函数参数



活动12 讨论例 6.14 有什么需要改进的地方,并用得出的结论或者发现的方法改进例 6.14,使之更完善。

在例 6.14 中我们使用数组元素作为函数参数,我们发现该例的数组赋初值和输出都在 **main** 函数中进行,这是因为 **main** 函数传给 **nzp** 函数的只是单个元素。因此主函数看起来不简洁,而实现该功能的函数又显得太简单,但如果数组元素为 100 个或者问题变得更复杂的话,每次只传一个数组元素的做法显然不理想。

我们的设想是,可以将整个数组作为参数传到函数里面进行操作,这样就能解决上述问题并使程序简洁,可维护性高。C语言系统其实已经帮我们设计好了这个功能,因此可以将例 6.14 改进如下:

【例 6.15】 数组名做函数参数。

【程序】

```
void nzp(int a[5])
{
   int i;
   printf("\nvalues of array a are:\n");
   for(i=0;i<5;i++)
   {
        if(a[i]<0) a[i]=0;
        printf("%d",a[i]);
   }
}
main()
{
   int b[5],i;
   printf("\ninput 5 numbers:\n");
   for(i=0;i<5;i++)
        scanf("%d",&b[i]);
   printf("initial values of array b are:\n");
   for(i=0;i<5;i++)
```

更多高清PDF图书 www.gqpdf.com

```
        printf("%d",b[i]);
    nzp(b);
    printf("\nlast values of array b are:\n");
    for(i=0;i<5;i++)
        printf("%d",b[i]);
}
```

【分析】 例6.15中函数nzp的形参为整数组a,长度为5。主函数中实参数组b也为整型,长度也为5。在主函数中首先输入数组b的值,然后输出数组b的初始值,然后以数组名b为实参调用nzp函数。在nzp函数中,按要求把负值单元清0,并输出形参数组a的值。返回主函数之后,再次输出数组b的值。从运行结果可以看出,数组b的初值和终值是不同的,数组b的终值和数组a是相同的。这说明实参形参为同一数组,它们的值同时得以改变。

(1)用数组名作函数参数与用数组元素作实参有以下几点不同:

①用数组元素作实参时,只要数组类型和函数的形参变量的类型一致,那么作为下标变量的数组元素的类型也和函数形参变量的类型是一致的。因此,并不要求函数的形参也是下标变量。换句话说,对数组元素的处理是按普通变量对待的。用数组名作函数参数时,则要求形参和相对应的实参都必须是类型相同的数组,都必须有明确的数组说明。当形参和实参二者不一致时,即会发生错误。

②在普通变量或下标变量作函数参数时,形参变量和实参变量是由编译系统分配的两个不同的内存单元。在函数调用时发生的值传送是把实参变量的值赋予形参变量。在用数组名作函数参数时,不是进行值的传送,即不是把实参数组的每一个元素的值都赋予形参数组的各个元素。因为实际上形参数组并不存在,编译系统不为形参数组分配内存。那么,数据的传送是如何实现的呢?在我们曾介绍过,数组名就是数组的首地址。因此在数组名作函数参数时所进行的传送只是地址的传送,也就是说把实参数组的首地址赋予形参数组名。形参数组名取得该首地址之后,也就等于有了实在的数组。实际上是形参数组和实参数组为同一数组,共同拥有一段内存空间。

(2)用数组名作为函数参数时还应注意以下几点:

①形参数组和实参数组的类型必须一致,否则将引起错误。

②形参数组和实参数组的长度可以不相同,因为在调用时,只传送首地址而不检查形参数组的长度。当形参数组的长度与实参数组不一致时,虽不至于出现语法错误(编译能通过),但程序执行结果将与实际不符,这是应予以注意的。

③在函数形参表中,允许不给出形参数组的长度,或用一个变量来表示数组元素的个数。

例如,可以写为:

```
void nzp(int a[])
```

或写为:

```
void nzp(int a[],int n)
```

其中形参数组a没有给出长度,而由n值动态地表示数组的长度。n的值由主调函数

高清PDF原创 www.gqpdf.com

的实参进行传送。

④多维数组也可以作为函数的参数。在函数定义时对形参数组可以指定每一维的长度,也可省去第一维的长度。因此,以下写法都是合法的。

```
intMA(int a[3][10])
```

或

```
intMA(int a[][10])
```

即时练习

使用数组名做函数参数从 main 函数得到 4 个一维数组,每个一维数组长度为 5,存放整型数值,在函数中将这 4 个一维数组组成一 4 行 5 列的二维数组并求它的转置矩阵,将结果打印出来。

6.5 本章小结

具有相同类型的若干变量按有序的形式组织起来,这些按序排列的同类数据元素的集合称为数组。数组按照维数可以分为一维、二维甚至 n 维。数组可以在定义时赋初值也可以定义后再赋值,数组必须逐个元素引用。字符数组是用来存放字符变量的数组,一个字符数组相当于一个字符串,字符串有常用的处理函数。数组除了普通的用法外还可以作为函数参数使用,有数组元素做函数参数和数组名做函数参数两种方式。

课后练习

1.输入 20 个数并进行排序后输出。

2.求一个 3×3 矩阵对角线元素之和。

3.打印杨辉三角。

4.编写一函数,将两个字符串 s1 和 s2 连接起来,不要使用 strcat 函数。

5.有一篇文章,共 10 行文字,每行有 80 个字符。要求分别统计出其中英文大写字母、小写字母、数字、空格以及其他字符的个数。

6.编写一函数,查找字符串 s2 在字符串 s1 中的位置,并返回该位置(位置从 1 开始,即第 1 个字符,比如 s2 从第 5 个字符开始则返回 5),如果找不到则返回 0,如果 s2 在 s1 中有多个位置则返回第一个位置。

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

第 7 章

指　针

【本章主要内容】

指针是C语言中广泛使用的一种数据类型。运用指针编程是C语言最主要的风格之一。利用指针变量可以表示各种数据结构;能很方便地使用数组和字符串;并能像汇编语言一样处理内存地址,从而编出精练而高效的程序。这对设计系统软件是很必要的。掌握指针的应用,可以使程序简洁、紧凑、高效。指针极大地丰富了C语言的功能。学习指针是学习C语言最重要的一环,能否正确理解和使用指针是我们是否掌握C语言的一个标志。同时,指针也是C语言中最为困难的一部分,在学习中除了要正确理解基本概念,还必须要多编程,上机调试。

【本章要求】

(1)掌握地址与指针变量的概念,地址运算符与指针运算符。

(2)掌握一维数组和字符串的地址以及指向变量、数组、字符串、函数的指针变量的定义。通过指针引用以上各类型数据。

(3)掌握用指针作函数参数。

(4)了解返回指针值的函数。

(5)了解多维指针。

7.1　指针和地址的概念

在计算机中,所有的数据都是存放在存储器中的。一般把存储器中的一个字节称为一个内存单元,不同的数据类型所占用的内存单元数不等,如整型变量占2个单元,字符型变量占1个单元等,前面章节已有详细的介绍。为了正确地访问这些内存单元,必须为每个内

高清PDF原创 www.gqpdf.com

存单元编号。根据每个内存单元的编号即可准确地找到该内存单元。内存单元的编号也叫做地址。既然根据内存单元的编号或地址就可以找到所需的内存单元,所以通常也把这个地址称为指针。内存单元的指针和内存单元的内容是两个不同的概念。可以用一个通俗的例子来说明它们之间的关系。我们到银行去存取款时,银行工作人员将根据我们的账号去找我们的存款单,找到之后在存单上写入存款、取款的金额。在这里,账号就是存单的指针,存款数是存单的内容。对于一个内存单元来说,单元的地址即为指针,其中存放的数据才是该单元的内容。在 C 语言中,允许用一个变量来存放指针,这种变量称为指针变量。因此,一个指针变量的值就是某个内存单元的地址或称为某内存单元的指针。

在图 7.1 中,设有字符变量 C,其内容为“K”(ASCII 码为十进制数 75),C 占用了 011A 号单元(地址用十六进数表示)。设有指针变量 P,内容为 011A,这种情况我们称为 P 指向变量 C,或说 P 是指向变量 C 的指针。

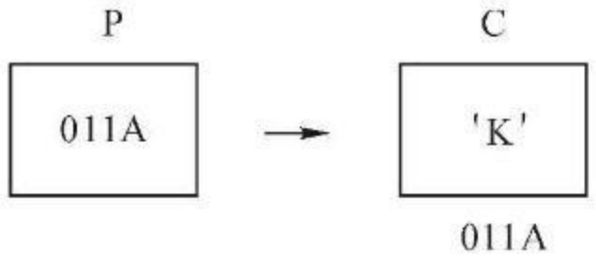


图 7.1 指针与单元地址

严格地说,一个指针是一个地址,是一个常量。而一个指针变量却可以被赋予不同的指针值,是变量。但常把指针变量简称为指针。为了避免混淆,我们约定:“指针”是指地址,是常量,“指针变量”是指取值为地址的变量。定义指针的目的是为了通过指针去访问内存单元。

既然指针变量的值是一个地址,那么这个地址不仅可以是变量的地址,也可以是其他数据结构的地址。在一个指针变量中存放一个数组或一个函数的首地址有何意义呢?因为数组或函数都是连续存放的。通过访问指针变量取得了数组或函数的首地址,也就找到了该数组或函数。这样一来,凡是出现数组,函数的地方都可以用一个指针变量来表示,只要对指针变量中赋予数组或函数的首地址即可。这样做,将会使程序的概念十分清楚,程序本身也精练,高效。在 C 语言中,一种数据类型或数据结构往往都占有一组连续的内存单元。用“地址”这个概念并不能很好地描述一种数据类型或数据结构,而“指针”虽然实际上也是一个地址,但它却是一个数据结构的首地址,“指向”一个数据结构,因而概念更为清楚,表示更为明确。这也是引入“指针”概念的一个重要原因。

7.2 变量的指针和指向变量的指针变量

在 C 语言中,一个变量的地址称为该变量的“指针”,通过它能找到以它为地址的内存单元。如果有一个变量专门用来存放另一变量的地址,则它称为“指针变量”。指针变量的值(即指针变量中存放的值)是指针(地址)。为了表示指针变量和它所指向的变量之间的联系,在程序中用“*”符号表示“指向”。例如,i_pointer 代表指针变量,*i_pointer 是

更多高清PDF图书 www.gqpdf.com

i_pointer 所指向的变量,见图7.2。

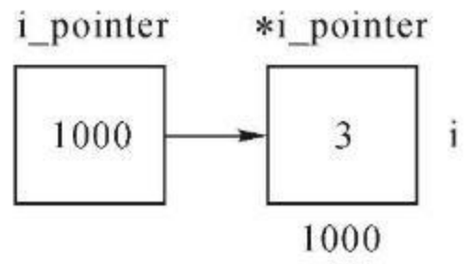


图7.2　指针和变量的关系

*i_pointer 代表一个变量,和变量 i 意义相同。下面两个语句作用相同:

```
① i=3;
② *i_pointer=3;
```

第②个语句的含意是将3赋给指针变量 i_pointer 所指向的变量。

7.2.1　指针变量的定义



活动1　如何定义一个指针变量。

例如:

```
int i, j;
int *pointer_1, *pointer_2;
```

第1行定义了两个整型变量 i 和 j,第2行定义了两个指针变量:pointer_1 和 pointer_2,它们是指向整型变量的指针变量。左端的 int 是在定义指针变量时必须指定的“基类型”。

用赋值语句使一个指针变量指向一个变量(将一个变量的地址赋给一个指针变量):

```
pointer_1=&i;
pointer_2=&j;
```

将变量 i 的地址存放到指针变量 pointer_1 中,pointer_1“指向”了变量 i。

将变量 j 的地址存放到指针变量 pointer_2 中,pointer_2“指向”了变量 j。

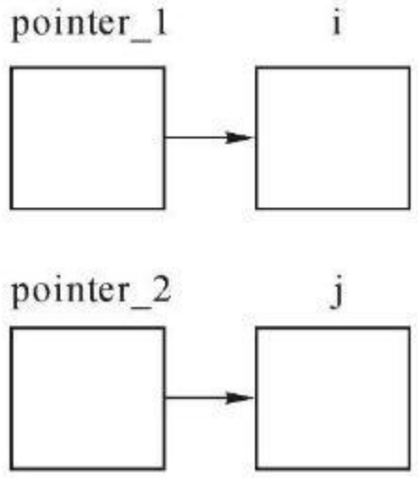


图7.3　使一个指针变量指向一个整型变量

通过上面的例子我们可以看出,对指针变量的定义包括三个内容:

(1)指针类型说明,即定义变量为一个指针变量;

高清PDF原创 www.gqpdf.com

(2)指针变量名;

(3)变量值(指针)所指向的变量的数据类型。

定义指针变量的一般形式为:

基类型 * 指针变量名

指针变量的基类型用来指定该指针变量可以指向的变量的类型。

在定义指针变量时注意两点:

(1)* 表示这是一个指针变量,变量名即为定义的指针变量名,类型说明符表示本指针变量所指向的变量的数据类型。

例如:

```
int *p1;
```

表示 p1 是一个指针变量,它的值是某个整型变量的地址。或者说 p1 指向一个整型变量。至于 p1 究竟指向哪一个整型变量,应由向 p1 赋予的地址来决定。

(2)在定义指针变量时必须指定其基类型。

例如:

```
int *p2;          /*p2是指向整型变量的指针变量*/
float *p3;        /*p3是指向浮点变量的指针变量*/
char *p4;         /*p4是指向字符变量的指针变量*/
```

一个指针变量只能指向同类型的变量,如 p3 只能指向浮点变量,不能时而指向一个浮点变量,时而又指向一个字符变量。

7.2.2 指针变量的引用

指针变量同普通变量一样,使用之前不仅要定义说明,而且必须赋予具体的值。未经赋值的指针变量不能使用,否则将造成系统混乱,甚至死机。指针变量的赋值只能赋予地址,决不能赋予任何其他数据,否则将引起错误。在 C 语言中,变量的地址是由编译系统分配的,对用户完全透明,用户不知道变量的具体地址。

两个有关的运算符:

(1)& :地址运算符(取址运算符)。&a 为变量 a 的地址。

(2)* :指针运算符("间接访问"运算符)。* p 为指针变量 p 所指向的存储单元。

C 语言中提供了地址运算符 & 来表示变量的地址。

其一般形式为:

& 变量名;

如 &a 表示变量 a 的地址,&b 表示变量 b 的地址。变量本身必须预先说明。

设有指向整型变量的指针变量 p,如要把整型变量 a 的地址赋予 p 可以有以下两种方式:

(1)指针变量初始化的方法:

```
int a;
int *p=&a;
```

更多高清PDF图书 www.gqpdf.com

(2)赋值语句的方法：

```
int a;
int * p;
p=&a;
```

不允许把一个数赋予指针变量，故下面的赋值是错误的：

```
int * p;
p=1000;
```

被赋值的指针变量前不能再加“*”说明符，如写为 * p=&a 也是错误的。

活动2　观察例7.1中指针变量所指向的整形变量单元，写出程序的输出结果，并区分程序中两处出现 * pointer_1 和 * pointer_2 的不同含义。

【例7.1】 通过指针变量访问整型变量。

【程序】

```
#include<stdio.h>
voidmain()
{
  int a, b;
  int * pointer_1, * pointer_2;
  a=100;
  b=10;
  pointer_1=&a;     /* 把变量 a的地址赋给 pointer_1 */
  pointer_2=&b;     /* 把变量 b的地址赋给 pointer_2 */
  printf("%d,%d\n", a, b);
  printf("%d,%d\n", * pointer_1, * pointer_2);     /* 输出指针变量指向的变量的值 */
  printf("%d,%d\n", pointer_1, pointer_2);     /* 输出指针变量指向的变量的地址 */
}
```

高清PDF原创 www.gqpdf.com

【分析】 通过活动2我们可以看出：

(1)在程序开头处定义了两个指针变量 pointer_1 和 pointer_2，规定它们可以指向整型变量。程序第8、9行的作用使 pointer_1 指向 a，pointer_2 指向 b，如图7.4所示。此时 pointer_1 的值为 &a(即 a 的地址)，pointer_2 的值为 &b。

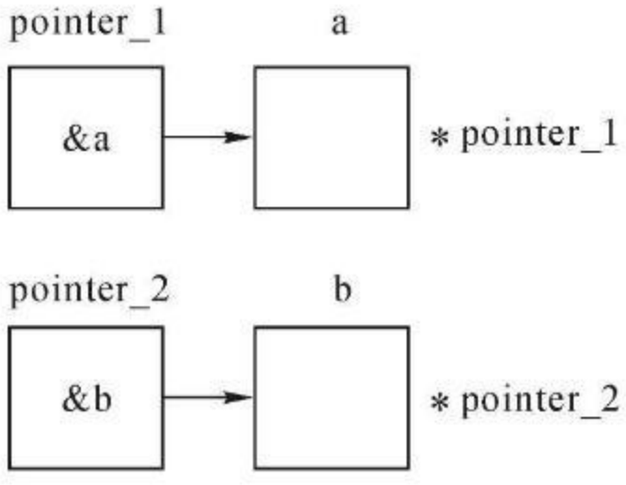


图7.4　指针变量的指向

(2)第 11 行的 * pointer_1 和 * pointer_2 就是变量 a 和 b。1、2 两个 printf 函数作用相同的。

(3)程序中有两处出现 * pointer_1 和 * pointer_2,它们表示的是不同的含义:程序第 5 行的 * pointer_1 和 * pointer_2 表示定义两个指针变量 pointer_1, pointer_2。它们前面的“ * ”只是表示该变量是指针变量;程序第 11 行 printf 函数中的 * pointer_1 和 * pointer_2 则代表变量,即 pointer_1 和 pointer_2 所指向的变量。

通过上面的学习,对关于“&”和“ * ”运算符的问题进行思考:

(1)如果已经执行了“pointer_1= &a;”语句,则 & * pointer_1 是什么含义?

(2) * &a 含义是什么?

(3)(pointer_1)++和 pointer_1++的区别?

假设已执行了“pointer_1= &a;”,则:

(1) & * pointer_1 的含义:先进行 * pointer_1 的运算,它就是变量 a,再执行 & 运算。因此,& * pointer_1 与 &a 等价,即变量 a 的地址(即指针变量 pointer_1)。

(2) * &a 的含义:先进行 &a 运算,得 a 的地址,再进行 * 运算,即 &a 所指向的变量,* &a 和 * pointer_1 的作用是一样的(即变量 a),见图 7.5。

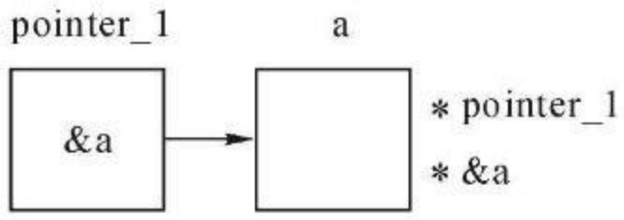


图 7.5 * &a 和 * pointer_1 的作用一样

(3)(* pointer_1)++相当于 a++。括号是必要的,如果没有括号,就成为了 * pointer_1++,++和 * 为同一优先级别,而结合方向为自右而左,因此它相当于 * (pointer_1++)。由于++在 pointer_1 的右侧,是“后加”,因此先对 pointer_1 的原值进行 * 运算,得到 a 的值,然后使 pointer_1 的值改变,这样 pointer_1 不再指向 a 了。

【例 7.2】 输入 a 和 b 两个整数,按先大后小的顺序输出 a 和 b。

【程序】

```
#include<stdio.h>
voidmain()
{
   int *p1, *p2, *p, a, b;
   scanf("%d%d", &a, &b);
   p1=&a;
   p2=&b;
   printf("a=%d, b=%d\n", a, b);
   printf("*p1=%d, *p2=%d\n", *p1, *p2);
   printf("p1=%d, p2=%d\n", p1, p2);
   if(a< b)     /* p1 和 p2 交换所指向的变量 */
   {
       p=p1;
       p1=p2;
```

更多高清PDF图书 www.gqpdf.com

```
        p2=p;
    }
    printf("a=%d,b=%d\n",a,b);
    printf("max(*p1)=%d,min(*p2)=%d\n",*p1,*p2);
    printf("p1=%d,p2=%d\n",p1,p2);
}
```

【分析】 当输入 a=5，b=9 时，由于 a＜b，将 p1 和 p2 交换。交换前的情况见图 7.6（a），交换后见图 7.6（b）。

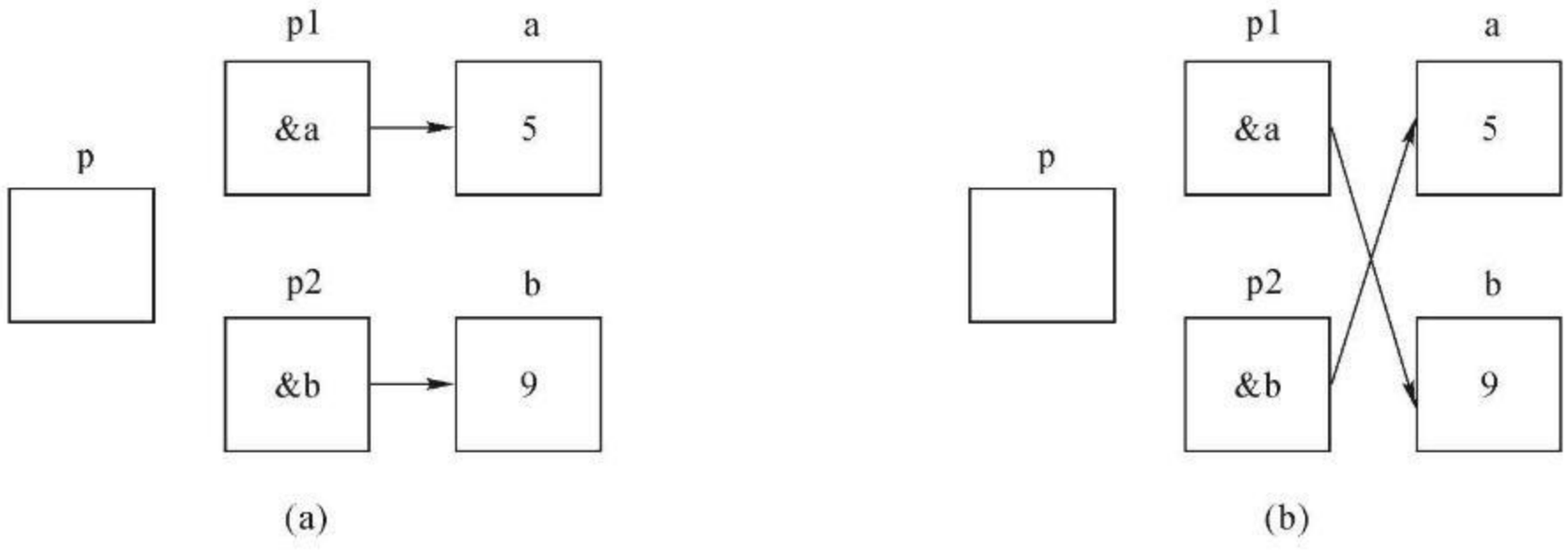


图 7.6　例 7.2 的示意

这个问题的算法是不交换整型变量的值，而是交换两个指针变量的值（即 a 和 b 的地址）。

7.2.3　指针变量作为函数参数

通过前面知识的学习，我们了解了函数的参数可以是整型、实型、字符型等数据，除此之外它还可以是指针类型。它的作用是将一个变量的地址传送到另一个函数中。

活动 3　回忆例 7.2，尝试使用函数调用的方式写出该程序，而且用指针类型的数据作为函数参数。

【例 7.3】 输入 a 和 b 两个整数，按先大后小的顺序输出 a 和 b。用指针类型的数据作函数参数。

【程序】

```
#include<stdio.h>
void swap(int *p1,int *p2)
{
    int temp;
    /* 交换指针变量 p1 和 p2 所指向的变量的值 */
    temp=*p1;
    *p1=*p2;
    *p2=temp;
}
```

```
voidmain()
{
   int a, b;
   int * pointer_1, * pointer_2;
   scanf("% d,% d", &a, &b);
   pointer_1=&a;
   pointer_2=&b;
   if(a< b)
      swap(pointer_1, pointer_2);
  printf("% d,% d\n", a, b);
}
```

【运行结果】

```
5,9↙
9,5
```

【分析】 对程序的说明:swap 是用户定义的函数,作用是交换两个变量(a 和 b)的值。swap 函数的两个形参 p1、p2 是指针变量。在函数调用时,将实参变量的值传送给形参变量,采取的是“值传递”方式,形参 p1 的值为 &a,p2 的值为 &b。这时 p1 和 pointer_1 都指向变量 a,p2 和 pointer_2 都指向变量 b。执行 swap 函数的函数体,使 * p1 和 * p2 的值互换,也就是使 a 和 b 的值互换,见图 7.7。

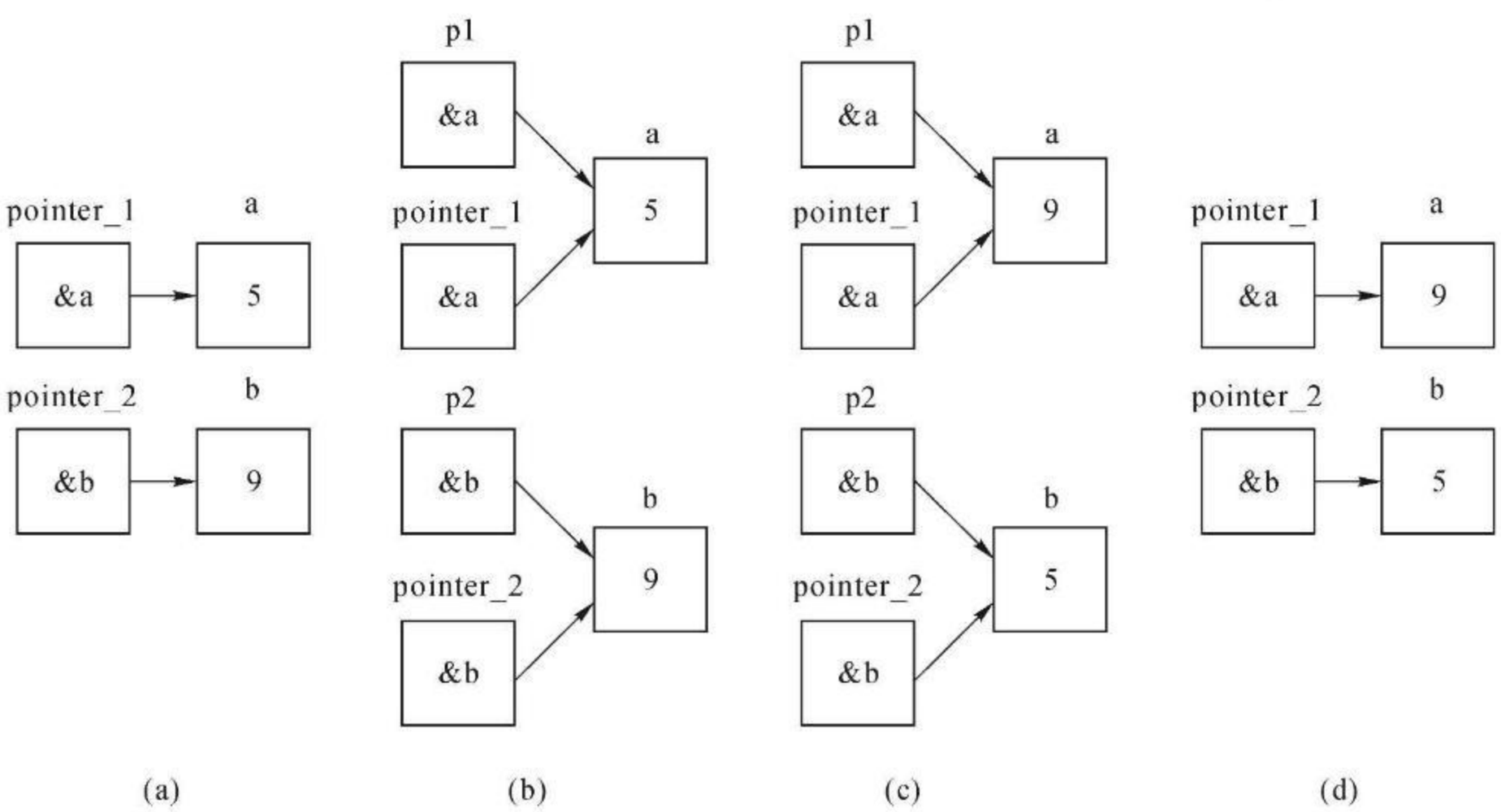


图 7.7 交换指针所指内容

注意交换 * p1 和 * p2 的值是如何实现的。

思考:如果 swap 函数写成如下形式,程序会按原来的目的运行吗?

```
void swap(int * p1, int * p2)
{
   int * temp;
```

更多高清PDF图书 www.gqpdf.com

```
    *temp=*p1;
    p1=*p2;
    p2=*temp;
}
```

我们会发现程序并没有按原目的运行,那么问题出在哪里呢? *p1 在这里就是 a,是整型变量。而 *temp 是指针变量 temp 所指向的变量。但 temp 中并无确定的地址值,它的值是不可预见的。*temp 所指向的单元也不是不可预见的。因此,对 *temp 赋值可能会破坏系统的正常工作状况。应该将 *p1 的值赋给一个整型变量,如例 7.3 中程序所示那样,用整型变量 temp 作为临时辅助变量实现 *p1 和 *p2 的交换。

从上面的例子中可以看到,在执行 swap 函数后,变量 a 和 b 的值改变了。那么,这个改变是怎么实现的呢?这个改变不是通过将形参值传回实参来实现的。

请大家考虑下面的函数能否实现 a 和 b 互换。

```
swap(int x, int y)
{
   int temp;
   temp=x;
   x=y;
   y=temp;
}
```

高清PDF原创 www.gqpdf.com

如果在 main 函数中用"swap(a,b);"调用 swap 函数,会有什么结果呢?见图 7.8。



图 7.8 普通变量的交换

在函数调用时,a 的值传送给 x,b 的值传送给 y。执行完 swap 函数后,x 和 y 的值是互换了,但 main 函数中的 a 和 b 并未互换。也就是说,由于"单向传递"的"值传递"方式,形参值的改变无法传给实参。所以为了使在函数中改变了的变量的值能够被 main 函数所用,不能采取上述把要改变值的变量作为参数的办法,而应该用指针变量作为函数参数,在函数执行过程中使指针变量所指向的变量值发生变化,函数调用结束后,这些变量值的变化依然保留下来,这样就实现了"通过调用函数使变量的值发生变化,在主调函数(如 main 函数)中使用这些改变了的值"的目的。

请注意,不能企图通过改变指针形参的值而使指针实参的值改变,如下面的程序。

```
#include<stdio.h>
void swap(int *p1,int *p2)
{
```

```
    int *p;
    p=p1;
    p1=p2;
    p2=p;
}
voidmain()
{
    inta,b;
    int *pointer_1,*pointer_2;
    scanf("%d,%d",&a,&b);
    pointer_1=&a;
    pointer_2=&b;
    if(a<b)
        swap(pointer_1,pointer_2);
    printf("\n%d,%d\n",*pointer_1,*pointer_2);
}
```

其中的问题在于不能实现如图 7.9 所示的第四步(d)。

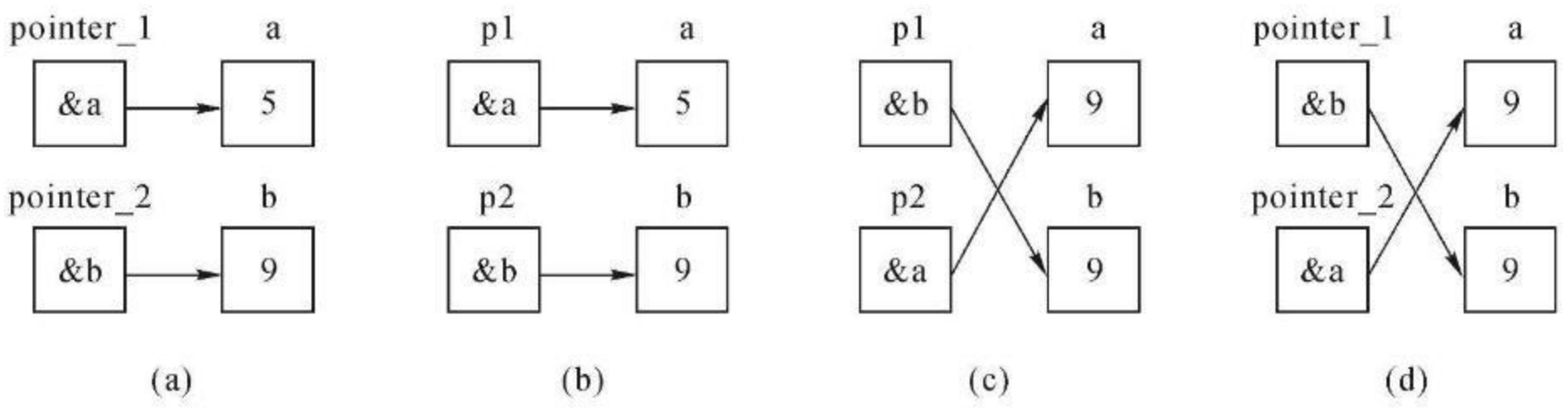


图 7.9 交换指针本身

如果想通过函数调用得到 n 个要改变的值,可以:

(1)在主调函数中设 n 个变量,用 n 个指针变量指向它们;

(2)然后将指针变量作为实参,将这 n 个变量的地址传给所调用的函数的形参;

(3)通过形参指针变量,改变该 n 个变量的值。

参考例 7.3,编写程序:输入 a、b、c 3 个整数,按大小顺序输出。

7.2.4 关于指针变量几个问题的进一步说明

指针变量可以进行某些运算,但其运算的种类是有限的。它只能进行赋值运算和部分算术运算及关系运算。

1.指针运算符

(1)取地址运算符 &:是单目运算符,其结合性为自右至左,其功能是取变量的地址。在 scanf 函数及前面介绍指针变量赋值中,我们已经了解并使用了 & 运算符。

(2)指针运算符 *:是单目运算符,其结合性为自右至左,用来表示指针变量所指的变

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

高清PDF原创 www.gqpdf.com

量。在*运算符之后跟的变量必须是指针变量。

需要注意的是指针运算符*和指针变量说明中的指针说明符*不是一回事。在指针变量说明中,“*”是类型说明符,表示其后的变量是指针类型。而表达式中出现的“*”则是一个运算符用以表示指针变量所指的变量。

例如:

```
voidmain()
{
 int a=5,*p=&a;
printf("%d",*p);
}
```

表示指针变量p取得了整型变量a的地址。“printf("%d",*p);”语句表示输出变量a的值。

2.指针变量的运算

(1)赋值运算:指针变量的赋值运算有以下几种形式。

①指针变量初始化赋值,前面已作介绍。

②把一个变量的地址赋予指向相同数据类型的指针变量。

例如:

```
 int a,*pa;
pa=&a;      /*把整型变量 a的地址赋予整型指针变量 pa*/
```

③把一个指针变量的值赋予指向相同类型变量的另一个指针变量。

例如:

```
 int a,*pa=&a,*pb;
pb=pa;      /*把 a的地址赋予指针变量 pb*/
```

由于pa,pb均为指向整型变量的指针变量,因此可以相互赋值。

④把数组的首地址赋予指向数组的指针变量。

例如:

```
 int a[5],*pa;
pa=a;
```

数组名表示数组的首地址,故可赋予指向数组的指针变量pa。

也可写为:

```
pa=&a[0];  /*数组第一个元素的地址也是整个数组的首地址,也可赋予 pa*/
```

当然也可采取初始化赋值的方法:

```
 int a[5],*pa=a;
```

⑤把字符串的首地址赋予指向字符类型的指针变量。

例如:

```
char * pc;
pc= "C Language";
```

或用初始化赋值的方法写为：

```
char * pc= "C Language";
```

这里应说明的是并不是把整个字符串装入指针变量，而是把存放该字符串的字符数组的首地址装入指针变量。在后面还将详细介绍。

⑥把函数的入口地址赋予指向函数的指针变量。

例如：

```
int ( * pf)();
pf= f;      /* f为函数名 * /
```

(2) 加减算术运算

对于指向数组的指针变量，可以加上或减去一个整数 n。设 pa 是指向数组 a 的指针变量，则 pa＋n，pa－n，pa＋＋，＋＋pa，pa－－，－－pa 运算都是合法的。指针变量加或减一个整数 n 的意义是把指针指向的当前位置(指向某数组元素)向前或向后移动 n 个位置。应该注意，数组指针变量向前或向后移动一个位置和地址加 1 或减 1 在概念上是不同的。因为数组可以有不同的类型，各种类型的数组元素所占的字节长度是不同的。如指针变量加 1，即向后移动 1 个位置表示指针变量指向下一个数据元素的首地址。而不是在原地址基础上加 1。例如：

```
int a[5], * pa;
pa= a;        /* pa指向数组 a,也是指向 a[0]* /
pa= pa+2;   /* pa指向 a[2],即 pa的值为 &pa[2]* /
```

指针变量的加减运算只能对数组指针变量进行，对指向其他类型变量的指针变量作加减运算是毫无意义的。

(3) 两个指针变量之间的运算：只有指向同一数组的两个指针变量之间才能进行运算，否则运算毫无意义。

①两指针变量相减：两指针变量相减所得之差是两个指针所指数组元素之间相差的元素个数。实际上是两个指针值(地址)相减之差再除以该数组元素的长度(字节数)。例如：pf1 和 pf2 是指向同一浮点数组的两个指针变量，设 pf1 的值为 2010H，pf2 的值为 2000H，而浮点数组每个元素占 4 个字节，所以 pf1－pf2 的结果为(2010H—2000H)/4＝4，表示 pf1 和 pf2 之间相差 4 个元素。两个指针变量不能进行加法运算。例如：pf1＋pf2 是什么意思呢？毫无实际意义。

②两指针变量进行关系运算：指向同一数组的两指针变量进行关系运算可表示它们所指数组元素之间的关系。

例如：

pf1＝pf2 表示 pf1 和 pf2 指向同一数组元素；

pf1＞pf2 表示 pf1 处于高地址位置；

pf1＜pf2 表示 pf2 处于低地址位置。

更多高清PDF图书 www.gqpdf.com

指针变量还可以与0比较。

设p为指针变量,则p=0表明p是空指针,它不指向任何变量;p !=0表示p不是空指针。

空指针是由对指针变量赋予0值而得到的。

例如:

```
#defineNULL 0
int *p=NULL;
```

对指针变量赋0值和不赋值是不同的。指针变量未赋值时,可以是任意值,是不能使用的。否则将造成意外错误。而指针变量赋0值后,则可以使用,只是它不指向具体的变量而已。

7.3　数组的指针和指向数组的指针变量

一个变量有一个地址,一个数组包含若干元素,每个数组元素都在内存中占用存储单元,它们都有相应的地址。所谓数组的指针是指数组的起始地址,数组元素的指针是数组元素的地址。

引用数组元素可以用下标法,也可以用指针法,即通过指向数组元素的指针找到所需的元素。使用指针法能使目标程序质量得到提高(占用内存少,运行速度快)。

7.3.1　指向数组元素的指针

一个数组是由连续的一块内存单元组成的。数组名就是这块连续内存单元的首地址。一个数组也是由各个数组元素(下标变量)组成的。每个数组元素按其类型不同占有几个连续的内存单元。一个数组元素的首地址也是指它所占有的几个内存单元的首地址。

定义指向数组元素的指针变量的方法,与定义指向变量的指针变量相同。

例如:

```
int a[10];      /* 定义 a为包含 10个整型数据的数组 */
int *p;         /* 定义 p为指向整型变量的指针变量 */
p=&a[0];        /* 把 a[0]元素的地址赋给指针变量 p */
```

C语言规定数组名代表数组的首地址(第0号元素的地址),见图7.10。因此,下面两个语句等价:

```
p=&a[0];
p=a;            /* 把 a数组的首地址赋给指针变量 p,而不是把数组 a各元素的值给 p* /
```

在定义指针变量时可以赋初值:

```
int *p=&a[0];
```

高清PDF原创 www.gqpdf.com

图 7.10　指向数组元素

等效于

```
int *p;
p=&a[0];      /* 注意,不是 *p=&a[0]; */
```

也等效于

```
int *p=a;
```

都是将 a 的首地址(即 a[0] 的地址)赋给指针变量 p(而不是赋给 *p)。

7.3.2　通过指针引用数组元素

假设 p 已定义为指针变量,并已给它赋了一个地址,使它指向某一个数组元素。如果有以下赋值语句:

```
*p=1;
```

表示对 p 当前所指向的数组元素赋予一个值(值为 1)。

C 语言规定:如果指针变量 p 已指向数组中的一个元素,则 p+1 指向同一数组中的下一个元素。

例如:数组元素是实型,每个元素占 4 个字节,则 p+1 意味着使 p 的值加 4 个字节,以使它指向下一元素。p+1 所代表的地址是 p+1×d,d 是一个数组元素所占的字节数。

活动 4　观察图 7.11,思考下列问题:

如果 p 的初值为 &a[0],则:(1) p+i 和 a+i;(2) *(p+i) 或 *(a+i)分别表示什么?

通过观察我们发现:

(1) p+i 和 a+i 就是 a[i] 的地址(指向 a 数组的第 i 个元素),见图 7.11。a 代表数组的首地址,a+i 也是地址,它的计算方法同 p+i。

(2) *(p+i) 或 *(a+i) 是 p+i 或 a+i 所指向的数组元素,即 a[i]。在编译时,对数组元素 a[i] 是处理成 *(a+i),即将数组的首地址加上相对位移量(偏移量)得到要找的元素的地址,然后找出该单元中的内容。[] 实际上是变址运算符,即将 a[i] 按 a+i 计算地址,然后找出此地址单元中的值。

更多高清PDF图书　www.gqpdf.com

(3) 指向数组的指针变量也可以带下标,如 p[i] 与 (p+i) 等价。

根据以上叙述,引用一个数组元素,可以用:

(1) 下标法,如 a[i] 形式;

(2) 指针法,如 *(p+i) 或 *(a+i)。其中 a 是数组名,p 是指向数组的指针变量,其初值 p=a。

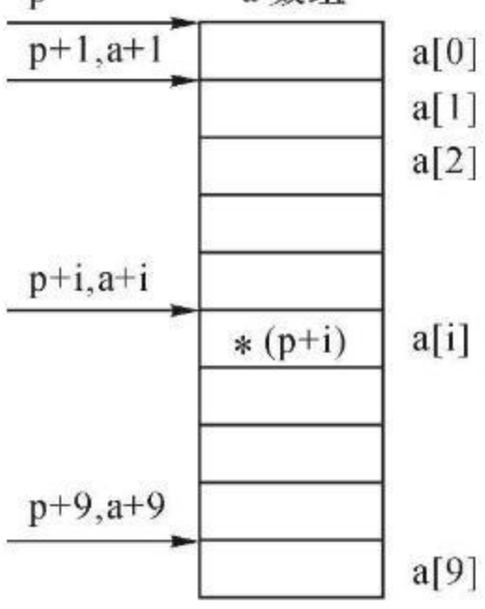


图 7.11　p+1=p+1×d

活动 5　分别用下标法和指针法输出数组中的全部元素,并比较两者的执行效率。

假设有一个整型数组 a,有 10 个元素。

(1) 下标法。

```
#include<stdio.h>
voidmain()
{
   int a[10];
   int i;
   for(i=0; i<10; i++)
       scanf("%d",&a[i]);
   printf("\n");
   for(i=0; i<10; i++)
       printf("%d ",a[i]);
   printf("\n");
}
```

(2) 通过数组名计算数组元素地址,找出元素的值。

```
#include<stdio.h>
voidmain()
{
   int a[10];
   int i;
   for(i=0; i<10; i++)
       scanf("%d",a+i);
```

高清PDF原创 www.gqpdf.com

```
    printf("\n");
    for(i=0; i<10; i++)
        printf("%d", *(a+i));
    printf("\n");
}
```

(3)用指针变量指向数组元素。

```
#include<stdio.h>
voidmain()
{
    int a[10];
    int *p, i;
    for(p=a; p<(a+10); p++)
    {
        printf("p=%d, *p=", p);
        scanf("%d", p);
    }
    printf("\n");
    for(p=a; p<(a+10); p++)
        printf("p=%d, *p=%d\n", p, *p);
    printf("\n");
}
```

以上程序的运行情况如下：

```
1 2 3 4 5 6 7 8 9 0↙
1 2 3 4 5 6 7 8 9 0
```

对三种方法的比较：

(1)第(1)和(2)种方法执行效率是相同的,即先计算元素地址,因此用第(1)和(2)种方法找数组元素费时较多。

(2)第(3)种方法比(1)和(2)法快,用指针变量直接指向元素,不必每次都重新计算地址,像 p++这样的自加操作是比较快的。这种有规律地改变地址值(p++)能大大提高执行效率。

(3)用下标法比较直观,能直接知道是第几个元素。用地址法或指针变量的方法不直观,难以很快地判断出当前处理的是哪一个元素。

在使用指针变量时,有几个问题要注意：

(1)指针变量可以实现使本身的值改变。例如,上述第(3)种方法是用指针变量 p 来指向元素,用 p++使 p 的值不断改变,这是合法的。如果不用 p 而使 a 变化(例如,用 a++)是无法实现的。因为 a 是数组名,它是数组的首地址,它的值在程序运行期间是固定不变的,是常量。

(2)要注意指针变量的当前值。请看下面的程序,并找出下面程序中的错误。

【例 7.4】 通过指针变量输出 a 数组的 10 个元素。

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

【程序】

```
#include<stdio.h>
voidmain()
{
  int *p, i, a[10];
  p=a;
  printf("%d\n", p);
  for(i=0; i<10; i++)
     scanf("%d", p++);
  printf("%d\n", p);
  for(i=0; i<10; i++, p++)
     printf("p=%d, *p=%d\n", p, *p);
}
```

【分析】 通过程序运行的结果,我们发现输出的数值显然不是 a 数组中各元素的值。原因是指针变量的初始值为 a 数组的首地址(见图 7.12 中的①),但经过第一个 for 循环读入数据后,p 已指向 a 数组的末尾(见图 7.12 中②)。因此,在执行第二个 for 循环时,p 的起始值已不是 &a[0]了,而是 a+10。

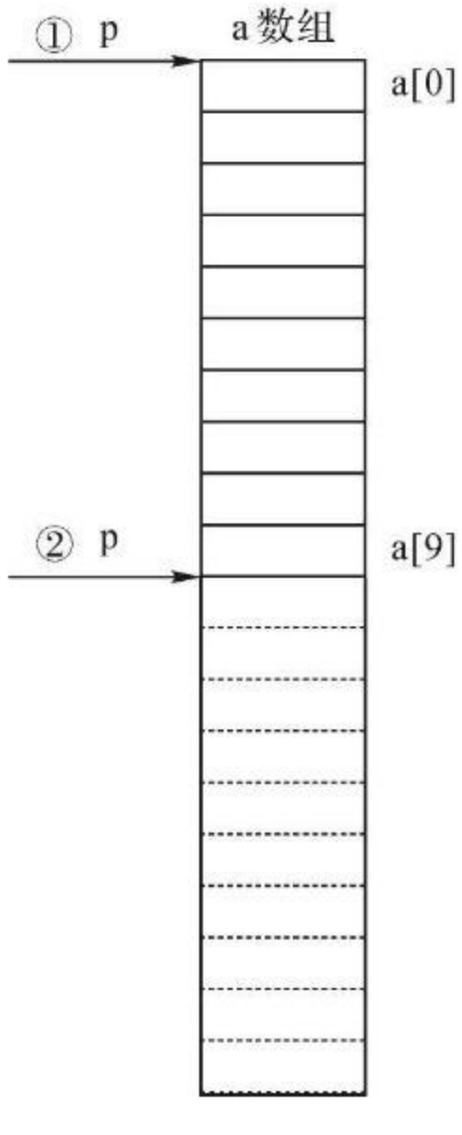


图 7.12 p 指向 a+10

解决办法:在第二个 for 循环之前加一个赋值语句,使 p 的初始值回到 &a[0]。

```
#include<stdio.h>
voidmain()
{
  int *p, i, a[10];
```

高清PDF原创 www.gqpdf.com

```
    p=a;
    printf("%d\n",p);
    for(i=0;i<10;i++)
        scanf("%d",p++);
    printf("%d\n",p);
    p=a;/* 在第二个 for循环之前加一个赋值语句,使 p的初始值回到&a[0]。 */
    for(i=0;i<10;i++,p++)
        printf("p=%d,*p=%d\n",p,*p);
}
```

虽然定义数组时指定它包含 10 个元素,用 p 指向当前的数组元素。但是实际上指针变量 p 可以指向数组以后的内存单元。如果引用数组元素 a[10],系统把它按 *(a+10) 处理,即先找出 (a+10) 的值(是一个地址),然后找出它指向的单元的内容。这样做虽然是"合法"的(在编译时不出错),但应避免出现这样的情况,这会使程序得不到预期的结果。在使用指针变量指向数组元素时,应切实保证指向数组中有效的元素。

(3) 注意指针变量的运算。如果先使 p 指向数组 a(即 p=a),则:

①p++(或 p+=1),使 p 指向下一元素,即 a[1];p--(p-=1),使 p 指向上一元素。

②(*p)++表示 p 所指向的元素的值加 1;(*p)--表示 p 所指向的元素的值减 1。

③*p++,由于++和 * 同优先级,结合方向为自右而左,因此它等价于 *(p++)。如果 p 当前指向 a 数组中第 i 个元素,则:

*(p++)　相当于 a[i++],先对 p 进行 * 运算,再使 p 自加;

*(p--)　相当于 a[i--],先对 p 进行 * 运算,再使 p 自减;

*(++p)　相当于 a[++i],先使 p 自加,再对 p 进行 * 运算;

*(--p)　相当于 a[--i],先使 p 自减,再对 p 进行 * 运算。

例 7.4 最后一个程序中最后一个 for 语句:

```
for(i=0;i<10;i++,p++)
    printf("%d",*p);
```

可以改写为:

```
for(i=0;i<10,i++)
    printf("%d",*p++);
```

作用完全一样,都是先输出 *p 的值,然后使 p 值加 1。

将++和--运算符用于指针变量十分有效,可以使指针变量自动向前或向后移动,指向下一个或上一个数组元素。

例如,想输出 a 数组 100 个元素,可以用如下程序:

更多高清PDF图书 www.gqpdf.com

```
p=a;

while(p< a+100)
  printf("%d", *p++);
```

或

```
p=a;

while(p< a+100)
{
  printf("%d", *p);
  p++;
}
```

7.3.3　用数组名作函数参数



活动6　讨论当用数组名作参数时,如果形参数组中元素的值发生变化,实参数组元素的值是否也随之变化?

通过之前的学习,我们了解了数组名可以作为函数的形参和实参。如:

```
void f(int arr[], int n)
{
  ......
}
```

```
voidmain()
{
int array[10];
  ......
f(array, 10);
  ......
}
```

实参数组名代表该数组的首地址,形参是用来接收从实参传递过来的数组的首地址。因此,形参是一个指针变量,因为只有指针变量才能存放地址。实际上,C 语言在编译时都是将形参数组作为指针变量来处理的。

例如,函数 f 的形参是写成数组形式的:

```
void f(int arr[], int n)
```

但在编译时是将 arr 按指针变量处理的,相当于将函数 f 的首部写成:

```
void f(int * arr, int n)
```

以上两种写法是等价的。在调用该函数时,系统会建立一个指针变量 arr,用来存放从主调函数传递过来的实参数组的首地址。当 arr 接收了实参数组的首地址后,arr 就指向实参数组的开头,也就是指向 array[0]。因此,* arr 就是 array[0] 的值。arr+i 指向 array[i], *(arr+i)是 array[i] 的值。根据前面介绍过的知识,*(arr+i) 和 arr[i] 是无条件等价的。因此,在调用函数期间,arr[0] 和 * arr 以及 array[0] 都是数组 array 第0号元素的值,依此类推,arr[i], *(arr+i), array[i] 都是 array 数组第 i 号元素的值,见图7.13。

关于变量名与数据名作为函数参数的区别可见表7.1。

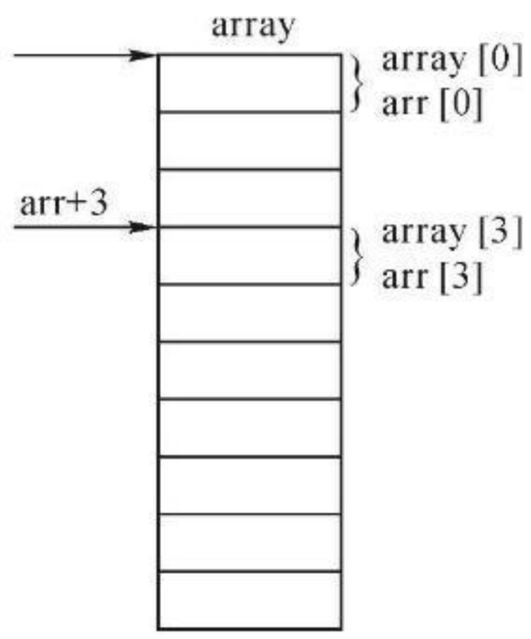


图 7.13 *(arr+i)和 arr[i]是无条件等价的

表 7.1 变量名和数组名分别作为函数参数的比较

实参类型	变量名	数组名
要求形参的类型	变量名	数组名或指针变量
传递的信息	变量的值	数组的首地址
通过函数调用能否改变实参的值	不能	能

说明:实参数组名代表一个固定的地址。形参数组名并不代表一个固定的地址,作为指针变量,在函数调用开始时,它的值等于实参数组起始地址,但在函数执行期间,它可以再被赋值。

【例 7.5】 将数组 a 中 n 个整数按相反顺序存放,见图 7.14。

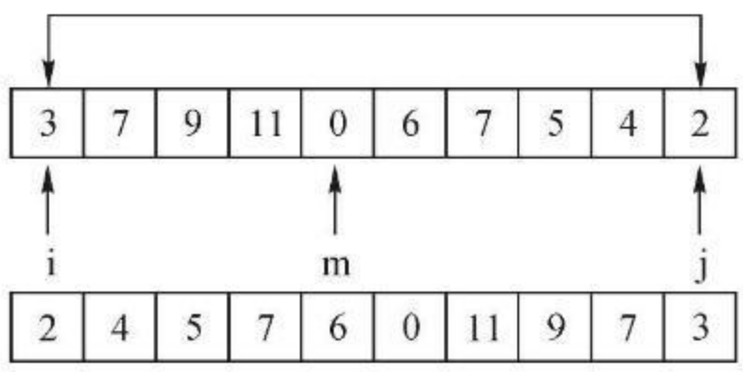


图 7.14 将数组 a 中 n 个整数按相反顺序存放

高清PDF原创 www.gqpdf.com

算法:将 a[0]与 a[n-1]对换,再将 a[1]与 a[n-2]对换……直到将 a[(n-1)/2]与 a[n-int((n-1)/2)-1]对换。

用循环处理此问题,设两个"位置指示变量"i 和 j,i 的初值为 0,j 的初值为 n-1。将 a[i]与 a[j]交换,然后使 i 的值加 1,j 的值减 1,再将 a[i]与 a[j]对换,直到 i=(n-1)/2 为止。

【程序】

```
#include<stdio.h>
void inv(int x[], int n)     /* 形参 x 是数组名 */
{
   int temp, i, j,m=(n-1)/2;
   for(i=0; i<=m; i++)
```

更多高清PDF图书 www.gqpdf.com

```
    {
        j=n-1-i;
        temp=x[i];
        x[i]=x[j];
        x[j]=temp;
    }
}
void main()
{
    int i,a[10]={3,7,9,11,0,6,7,5,4,2};
    printf("The original array:\n");
    for(i=0;i<10;i++)
        printf("%d",a[i]);
    printf("\n");
    inv(a,10);
    printf("The array has been inverted:\n");
    for(i=0;i<10;i++)
        printf("%d",a[i]);
    printf("\n");
}
```

【运行结果】

```
The original array:
3,7,9,11,0,6,7,5,4,2
The array has been inverted:
2,4,5,7,6,0,11,9,7,3
```

对这个程序作一些改动:将函数 inv 中的形参 x 改成指针变量。实参为数组名 a,即数组 a 的首地址,将它传给形参指针变量 x,这时 x 就指向 a[0]。x+m 是 a[m]元素的地址。设 i 和 j 以及 p 都是指针变量,用它们指向有关元素。i 的初值为 x,j 的初值为 x+n-1,见图7.15。使 *i 与 *j 交换就是使 a[i]与 a[j]交换。

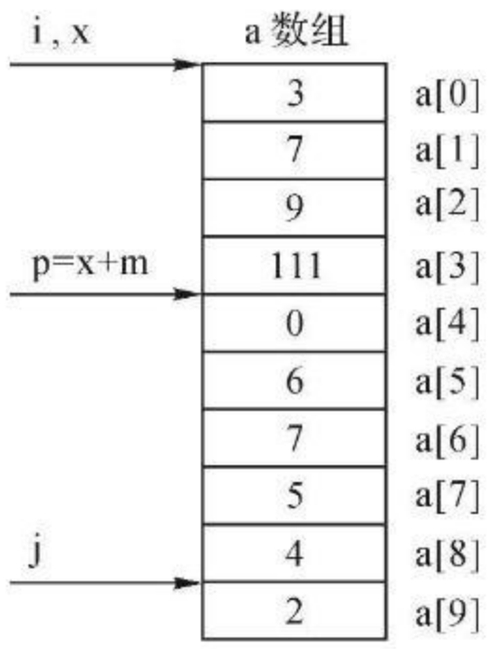


图7.15　x+m 是 a[m]元素的地址

```
void inv(int *x, int n)    /* 形参 x 为指针变量 */
{
  int *p, temp, *i, *j,m=(n-1)/2;
  i=x;
  j=x+n-1;
  p=x+m;
  for(; i<=p; i++, j--)
  {
      temp=*i;
      *i=*j;
      *j=temp;
  }
}
```

运行的情况和之前的程序是相同的。

【例 7.6】 从 10 个数中找出其中最大值和最小值。

调用一个函数只能得到一个返回值,为了能得到两个结果值,今用全局变量在函数之间“传递”数据。

【程序】

```
#include<stdio.h>
int max,min;    /* 全局变量 */
void max_min_value(int array[], int n)
{
  int *p, *array_end;
  array_end=array+n;
  max=min=*array;
  for(p=array+1; p<array_end; p++)
      if(*p>max)
          max=*p;
      else
          if(*p<min)
              min=*p;
}
void main()
{
  int i, number[10];
  printf("Enter 10 integer numbers:\n");
  for(i=0; i<10; i++)
      scanf("%d",&number[i]);
  max_min_value(number, 10);
  printf("\nmax=%d, min=%d\n",max,min);
}
```

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

【分析】 将函数 max_min_value 中求出的最大值和最小值放在 max 和 min 中。由于它们是全局变量,因此在主函数中可以直接使用。函数 max_min_value 中的语句:

```
max=min= *array;
```

array 是形参数组名,接收从实参传来的数组 number 的首地址。array 是形参数组的首地址,*array 相当于 *(array+0),即 array[0]。上述语句与下面语句等价:

```
max=min=array[0];
```

见图 7.16。在执行 for 循环时,p 的初值为 array+1,使 p 指向 array[1]。以后每次执行 p++,使 p 指向下一个元素。每次将 *p 和 max 与 min 比较,将大者放入 max,小者放入 min。

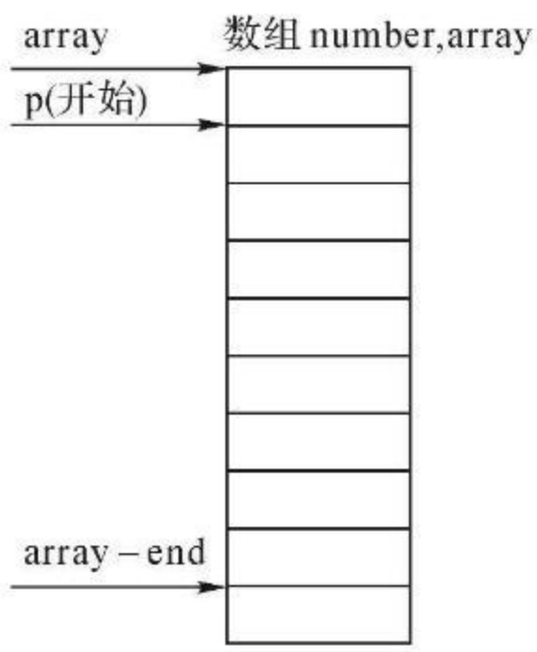


图 7.16　p 指向 array+1

实参和形参改用指针变量:

```
#include<stdio.h>
intmax,min;     /* 全局变量 */
voidmax_min_value(int *array, intn)
{
  int *p, *array_end;
  array_end=array+n;
  max=min= *array;
  for(p=array+1;p<array_end;p++)
      if(*p>max)
         max= *p;
      else
         if(*p<min)
            min= *p;
}
voidmain()
{
  int i,number[10], *p;
  p=number;     /* 使 p 指向 number 数组 */
  printf("Enter10 integernumbers:\n");
  for(i=0;i<10;i++,p++)
      scanf("%d",p);
  printf("The10 integernumbers:\n");
  for(p=number,i=0;i<10;i++,p++)
      printf("%d ",*p);
  p=number;
  max_min_value(p,10);     /* 实参用指针变量 */
  printf("\nmax=%d,min=%d\n",max,min);
}
```

高清PDF原创 www.gqpdf.com

归纳:如果有一个实参数组,想在函数中改变此数组的元素的值,实参与形参的表示形式有以下 4 种情况:

(1)形参和实参都用数组名。可以认为形参数组与实参数组共用内存单元,见图 7.17。

```
main()
{
 int a[10];
 …
 f(a, 10);
 …
}
```

```
f( int x[], int n)
    {
    …
    }
```

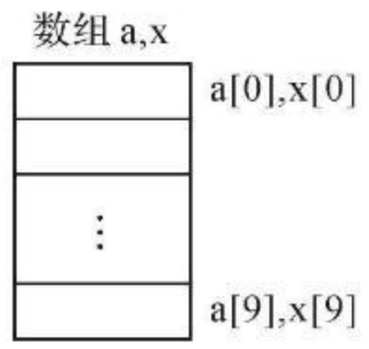


图 7.17 形参和实参都用数组名

(2)实参用数组名,形参用指针变量。实参 a 为数组名,形参 x 为指向整型变量的指针变量,函数开始执行时,x 指向 a[0],即 x=&a[0],见图 7.18。通过 x 值的改变,可以指向 a 数组的任一元素。

```
main()
{
 int a[10];
 …
 f(a, 10);
 …
}
```

```
f( int *x, int n)
    {
    …
    }
```

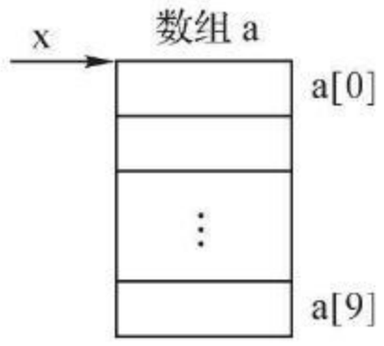


图 7.18 实参用数组名,形参用指针变量

(3)实参形参都用指针变量。实参 p 和形参 x 都是指针变量。先使实参指针变量 p 指向数组 a,p 的值是 &a[0]。然后将 p 的值传给形参指针变量 x,x 的初始值也是&a[0],见图 7.19。通过 x 值的改变可以使 x 指向数组 a 的任一元素。

```
main()
{
 int a[10], *p;
 p=a;
 …
 f(a, 10);
 …
}
```

```
f( int *x, int n)
    {
    …
    }
```

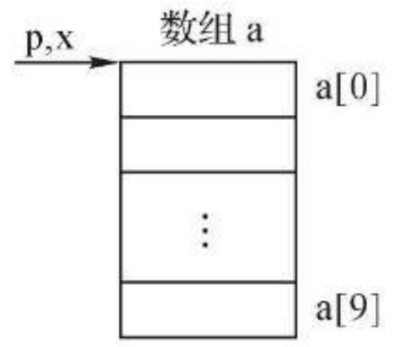


图 7.19 实参形参都用指针变量

(4)实参为指针变量,形参为数组名。实参 p 为指针变量,它使指针变量 p 指向 a[0],即 p=a 或 p=&a[0]。形参为数组名 x,将 a[0]的地址传给形参 x,使指针变量 x 指向 a[0](形参数组 x 取得 a 数组的首地址,x 数组和 a 数组共用同一段内存单元)。见图 7.20。在函数执行过程中可以使 x[i]值变化,它就是 a[i]。

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

```
main()
{
 int a[10], *p;
 p=a;
 …
 f(a, 10);
 …
}
```

```
f(int x[], int n)
    {
    …
    }
```

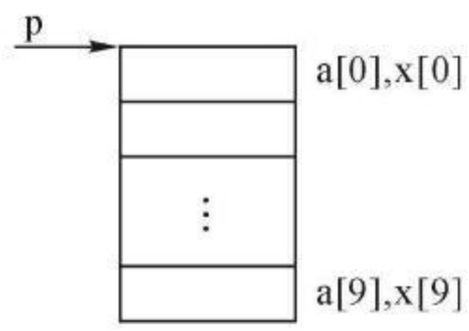


图 7.20 形参和实参都用数组名

以上四种方法,实质上都是地址的传递(即都是使用指针变量)。

【例 7.7】 用实参指针变量改写例 7.5。

【程序】

```
#include<stdio.h>
void inv(int *x, int n)
{
  int *p,m, temp, *i, *j;
  m=(n-1)/2;
  i=x;
  j=x+n-1;
  p=x+m;
  for(; i<=p; i++, j--)
  {
      temp=*i;
      *i=*j;
      *j=temp;
  }
}
voidmain()
{
  int i, arr[10], *p=arr;

  printf("The original array:\n");
  for(i=0; i<10; i++, p++)
      scanf("%d", p);
  printf("\n");
  p=arr;
  inv(p, 10);     /* 实参为指针变量 */
  printf("The array has been inverted:\n");
  for(p=arr; p<arr+10; p++)
      printf("%d ", *p);
  printf("\n");
}
```

人民邮电PDF原创 www.gqpdf.com

【分析】 注意:上面的 main 函数中的指针变量 p 是有确定值的。如果在 main 函数中不设数组,只设指针变量,就会出错。假如把程序修改如下:

```
voidmain()
{
   int i, * arr;
   printf("The original array:\n");
   for(i=0; i< 10; i++)
       scanf("%d", arr+ i);
   printf("\n");
   inv(arr, 10);      /* 实参为指针变量,但未被赋值 */
   printf("The array has been inverted:\n");
   for(i=0; i< 10; i++)
       printf("%d ", *(arr+ i));
   printf("\n");
}
```

编译时就会警告,原因是指针变量 arr 没有确定值,谈不上指向哪个变量。

注意:如果用指针变量作实参,必须先使指针变量指向一个已定义的数组。

【例 7.8】 用选择法对 10 个整数排序。

【程序】

```
#include<stdio.h>
void sort(int x[], int n)
{
   int i, j, k, t;
   for(i=0; i< n-1; i++)
   {
       k= i;
       for(j= i+1; j< n; j++)
           if(x[j]> x[k])
               k= j;
       if(k != i)
       {
           t=x[i];
           x[i]=x[k];
           x[k]= t;
       }
   }
}
voidmain()
{
   int * p, i, a[10];
   p= a;
```

更多高清PDF图书 www.gqpdf.com

```
for(i=0; i<10; i++)
    scanf("%d",p++);
p=a;
sort(p,10);
for(p=a, i=0; i<10; i++)
{
    printf("%d ", *p);
    p++;
  }
  printf("\n");
}
```

7.3.4　指向多维数组的指针和指针变量

1.多维数组的地址

设有一个二维数组 a,它有 3 行 4 列:

```
int a[3][4]={{1,3,5,7},{9,11,13,15},{17,19,21,23}};
```

a 是一个数组名。a 数组包含 3 行,即 3 个元素:a[0],a[1],a[2]。每一元素又是一个一维数组,包含 4 个元素(即 4 个列元素),例如,a[0]所代表的一维数组又包含 4 个元素:a[0][0],a[0][1],a[0][2],a[0][3],见图 7.21。

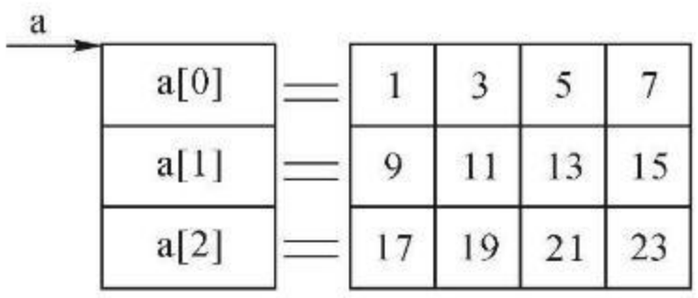


图 7.21　二维数的行和列元素

a 代表整个二维数组的首地址,也就是第 0 行的首地址。a+1 代表第 1 行的首地址。如果二维数组的首地址为 2000,则 a+1 为 2008,因为第 0 行有 4 个整型数据,因此 a+1 的含义是 a[1]的地址,即 a+4×2=2008。a+2 代表第 2 行的首地址,它的值是 2016,见图 7.22。

a[0],a[1],a[2]既然是一维数组名,而 C 语言又规定了数组名代表数组的首地址,因此 a[0]代表第 0 行一维数组中第 0 列元素的地址,即 &a[0][0]。a[1]的值是 &a[1][0],a[2]的值是 &a[2][0]。

第 i 行第 j 列元素的地址可以用 a[i]+j 来表示。此时"a[i]+j"中的 j 代表 j 个列元素的字节数,即 2 个字节。这是因为现在是在一维数组范围内讨论问题的,正如有一个一维数组 x,x+j 是其第 j 列元素地址一样。

a[i]和 *(a+i)等价。因此,a[i]+j 和 *(a+i)+j 的值都是 &a[i][j]。

用地址法表示 a[i][j]的值:既然 a[i]+j 和 *(a+i)+j 是 a[i][j]的地址,那么,*(a

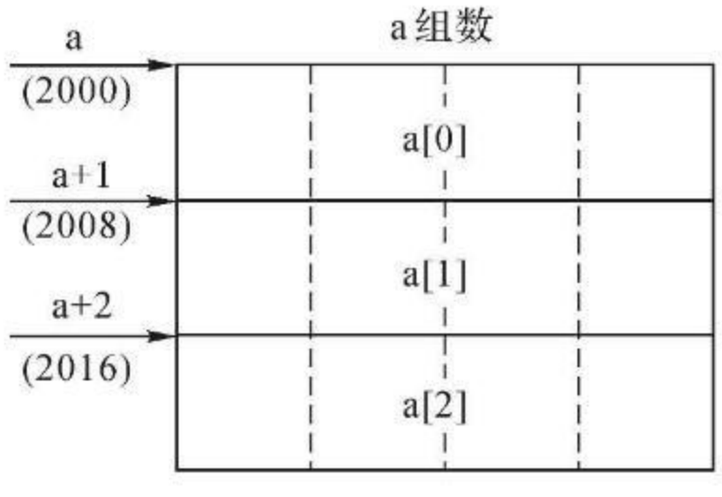


图 7.22　二维数组的行地址

[i]+j) 就是 a[i][j] 的值。同理，*(*(a+i)+j) 也是 a[i][j] 的值。

对 a[i] 的性质的进一步说明：a[i] 从形式上看是 a 数组中第 i 个元素。如果 a 是一维数组名，则 a[i] 代表 a 数组第 i 个元素所占的内存单元。a[i] 是有物理地址的，是占内存单元的。但如果 a 是二维数组，则 a[i] 是代表一维数组名。a[i] 本身并不占实际的内存单元，也不存放 a 数组中各个元素的值，只是一个地址(如同一个一维数组名 x 并不占内存单元而只代表地址一样)。具体的指向二维数组的指针的表示形式及含义见表 7.2。

表 7.2　指向二维数组的指针的表示形式及含义

表示形式	含　义
&a	指向二维数组的指针
a	二维数组名，指向一维数组 a[0]，即第 0 行的首地址
a[0]，*(a+0)，*a	第 0 行第 0 列元素的地址
a+i，&a[i]	第 i 行的首地址
a[i]，*(a+i)	第 i 行第 0 列元素的地址
a[i]+j，*(a+i)+j，&a[i][j]	第 i 行第 j 列元素的地址
(a[i]+j)，(*(a+i)+j)，a[i][j]	第 i 行第 j 列元素的值

a 和 a[0]的值虽然相同，但是由于指针的类型不同(a 是指向一维数组，a[0]指向 a[0][0]元素)。因此，对这些指针进行加 1 的运算，得到的结果是不同的。二维数组名是指向行的。因此 a+1 中的“1”代表一行中全部元素所占的字节数。一维数组名(如 a[0]，a[1])是指向列元素的。因此 a[0]+1 中的“1”代表一个元素所占的字节数。

在行指针前面加一个 * ，就转换为列指针。例如，a 和 a+1 是行指针，在它们前面加一个 * 就是 *a 和 *(a+1)，它们就成为列指针，分别指向 a 数组 0 行 0 列的元素和 1 行 0 列的元素。

在列指针前面加 & ，就成为行指针。例如 a[0]是指向 0 行 0 列元素的列指针，在它前面加一个 & ，得 &a[0]，由于 a[0]与 *(a+0)等价，因此 &a[0]与 & *a 等价，也就是与 a 等价，它指向二维数组的 0 行。

在二维数组中，a+i=a[i]=*(a+i)=&a[i]=&a[i][0]，即它们的地址值是相等的。

活动 7　通过下面的例子加强对指向二维数组的指针的认识。

【例 7.9】 输出二维数组有关的值。

更多高清PDF图书　www.gqpdf.com

【程序】

```
#include<stdio.h>
#define FORMAT "%d,%d\n"
void main()
{
  int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
  printf(FORMAT, a, *a);                          /* 0行首地址和0行0列元素地址 */
  printf(FORMAT, a[0], *(a+0));                   /* 0行0列元素地址 */
  printf(FORMAT,&a[0],&a[0][0]);                  /* 0行首地址和0行0元素地址 */
  printf(FORMAT, a[1], a+1);                      /* 1行0列元素地址和1行首地址 */
  printf(FORMAT,&a[1][0], *(a+1)+0);              /* 1行0列元素地址 */
  printf(FORMAT, a[2], *(a+2));                   /* 2行0列元素地址 */
  printf(FORMAT,&a[2], a+2);                      /* 2行首地址 */
  printf(FORMAT, a[1][0], *(*(a+1)+0));           /* 1行0列元素的值 */
}
```

【分析】 注意:a是二维数组名,代表数组的首地址,但是不能企图用 *a 来得到 a[0][0]的值。*a 相当于 *(a+0),即 a[0],它是第0行地址。a 是指向一维数组的指针,可理解为行指针,*a 是指向列元素的指针,可理解为列指针,指向0行0列元素,**a 是0行0列元素的值。同样,a+1 指向第1行首地址,但也不能企图用 *(a+1)得到 a[1][0]的值,而应该用 **(a+1)求 a[1][0]元素的值。

2.指向多维数组的指针变量

(1)指向数组元素的指针变量

【例7.10】 用指针变量输出数组元素的值。

【程序】

```
#include<stdio.h>
void main()
{
  int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
  int *p;
  for(p=a[0];p<a[0]+12;p++)
  {
      if((p-a[0])% 4 == 0)
          printf("\n");
      printf("%4d", *p);
  }
  printf("\n");
}
```

可将程序最后两个语句改为:

```
printf("addr=%o, value=%4d\n", p, *p);
```

高清PDF原创 www.gqpdf.com

【分析】 可以把 p 的值(即数组元素的地址)输出。

如果要输出某个指定的数组元素,则应事先计算该元素在数组中的相对位置(即相对于数组起始位置的相对位移量)。计算 a[i][j] 在数组中的相对位置的计算公式为 i * m+j(二维数组大小为 n×m)。如果开始时使指针变量 p 指向 a(即 a[0][0]),为了得到 a[i][j] 的值,可以用 *(p+i * n+m) 表示。a[i][j] 的地址为 a[0]+i * m+j。从图 7.23 可以看到在 a[i][j] 元素之前有 i 行元素(每行有 m 个元素),在 a[i][j] 所在行,a[i][j] 的前面还有 j 个元素,因此 a[i][j] 之前共有 i * m+j 个元素。

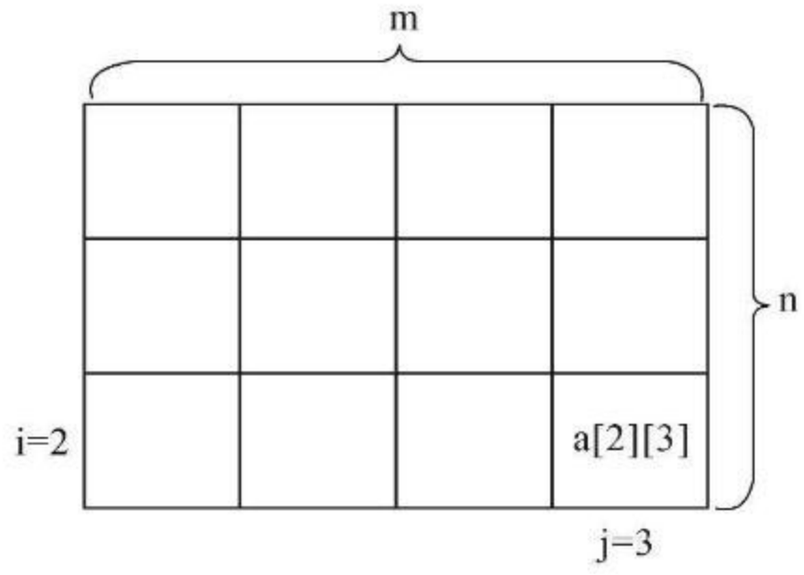


图 7.23 元素在数组中的相对位置

(2)指向由 m 个元素组成的一维数组的指针变量

使 p 不是指向整型变量,而是指向一个包含 m 个元素的一维数组。这时,如果 p 先指向 a[0](即 p=&a[0]),则 p+1 不是指向 a[0][1],而是指向 a[1],p 的增值以一维数组的长度为单位,见图 7.24。

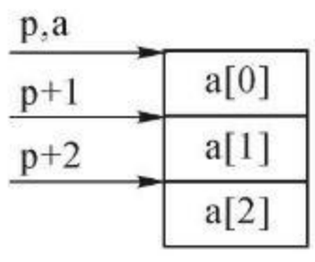


图 7.24 p 指向一个包含 m 个元素的一维数组

【例 7.11】 输出二维数组任一行任一列元素的值。

【程序】

```
#include<stdio.h>
voidmain()
{
  int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
  int(*p)[4],i,j;
  p=a;
  scanf("i=%d,j=%d",&i,&j);
  printf("a[%d,%d]=%d\n",i,j,*(*(p+i)+j));
}
```

【分析】 程序中“int(*p)[4]”表示 p 是一个指针变量,它指向包含 4 个元素的一维数

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

组。*p 两侧的括号不可缺少，如果写成 *p[4]，由于方括号 [] 运算级别高，因此 p 先与 [4] 结合，是数组，然后再与前面的 * 结合，*p[4] 是指针数组。

二者的比较：

```
① int a[4];        /* a有4个元素,每个元素为整型 */
② int (*p)[4];     /*  *p有4个元素,每个元素为整型* /
```

第②种形式 p 所指的对象是有 4 个整型元素的数组，即 p 是行指针，见图 7.25。此时 p 只能指向一个包含 4 个元素的一维数组，p 的值就是该一维数组的首地址。p 不能指向一维数组中的第 j 个元素。

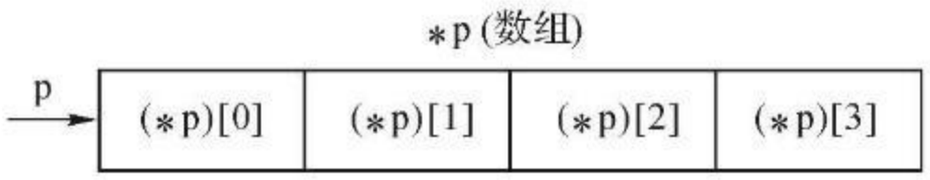


图 7.25　p 所指的对象是有 4 个整型元素的数组

程序中的 p+i 是二维数组 a 的第 i 行的地址(由于 p 是指向一维数组的指针变量，因此 p 加 1，就指向下一个一维数组)。见图 7.26。*(p+i)+j 是 a 数组第 i 行第 j 列元素地址，*(*(p+i)+j) 是 a[i][j] 的值。

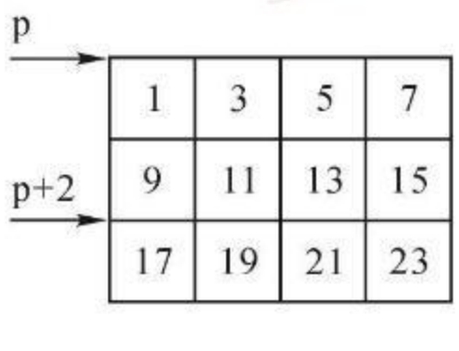


图 7.26　p+i 是二维数组 a 的第 i 行的地址

3.多维数组的指针作函数参数

用指针变量作形参接受二维实参数组名传递来的地址时，有两种方法：①用指向变量的指针变量；②用指向一维数组的指针变量。用指针变量作形参，可以允许数组的行数不同。

活动 8　利用多维数组的指针对学生成绩做统计。

【例 7.12】 有一个班，3 个学生，各学 4 门课，计算总平均分数，以及第 n 个学生的成绩。

用函数 average 求总平均成绩，用函数 search 找出并输出第 i 个学生的成绩。

【程序】

```
#include<stdio.h>
voidmain()
{
  void average(float *p, intn);
  void search(float (*p)[4], intn);
  float score[3][4] ={{65, 67, 70, 60},{80, 87, 90, 81},{90, 99, 100, 98}};
  average(*score, 12);     /* 求12个分数的平均分 */
```

```
    search(score, 2);     /* 求第 2 个学生成绩 */
}
void average(float *p, int n)
{
    float *p_end;
    float sum=0, aver;
    p_end=p+n-1;
    for(; p<=p_end; p++)
        sum= sum+(*p);
    aver= sum / n;
    printf("average=%5.2f\n", aver);
}
void search(float (*p)[4], int n)
{
    int i;
    printf("The score of No.%d are:\n", n);
    for(i=0; i< 4; i++)
        printf("%5.2f", (*(p+n)+ i));
    printf("\n");
}
```

【分析】 在函数 average 中形参 p 被声明为指向一个实型变量的指针变量。用 p 指向二维数组的各个元素，p 每加 1 就改为指向下一个元素，见图 7.27。相应的实参用 *score，即score[0]，是一个地址，指向 score[0][0] 元素。用形参 n 代表需要求平均值的元素的个数。函数 average 中的指针变量 p 指向 score 数组的某一元素(元素值为一门课的成绩)。

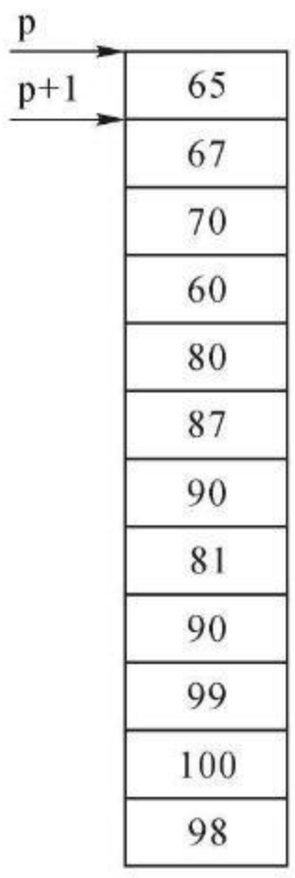


图 7.27　p 每加 1 就改为指向下一个元素

函数 search 的形参 p 是指向包含 4 个元素的一维数组的指针变量。实参传给形参 n 的值为 2，即找序号为 2 的学生的成绩(3 个学生的序号分别为 0、1、2)。函数调用开始时，

更多高清PDF图书 www.gqpdf.com

将实参 score 的值(代表该数组第 0 行首地址)传给 p。p＋n 是一维数组 score[n] 的首地址，*(p＋n)＋i 是 score[n][i] 的地址，*(*(p＋n)＋i) 是 score[n][i] 的值。现在n＝2，i 由 0 变到 3，for 循环输出 score[2][0] 到 score[2][3] 的值。

【例 7.13】 在例 7.12 的基础上，查找有一门以上课程不及格的学生，打印出他们的全部课程的成绩。

【程序】

```
#include <stdio.h>
voidmain()
{
  void search(float (*p)[4], intn);     /* 函数声明 */
  float score[3][4] ={{65, 67, 70, 60},{80, 87, 90, 81},{90, 99, 100, 98}};
  search(score, 3);
}
void search(float (*p)[4], intn)
{
  int i, j, flag;
  for(j=0; j< n; j++)
  {
      flag=0;
      for(i=0; i< 4; i++)
      {
          if(*(*(p+ j)+ i)< 60)
              flag=1;
          break;
      }
  }
  if(flag == 1)
  {
      printf("No.%d fails, his scores are:\n", j+1);
      for(i=0; i< 4; i++)
          printf("%5.1f", *(*(p+ j)+ i));
      printf("\n");
  }
}
```

【分析】 在函数 search 中，flag 是作为标志不及格的变量。先使 flag＝0，若发现某一学生有一门不及格，则使 flag＝1。最后用 if 语句检查 flag，如为 1，则表示该学生有不及格的纪录，输出其全部课程成绩。变量 j 代表学生号，i 代表课程号。

7.4　返回指针值的函数

一个函数可以返回一个整型值、实型值、字符型值等，也可以返回指针型的数据，即地

址。定义返回指针值的函数的一般形式为：

类型名 * 函数名(参数表)；

例如：

```
int * a(int x, int y);
```

a 是函数名，调用它以后能得到一个指向整型数据的指针(地址)。x，y 是函数 a 的形参，为整型。* a 两侧没有括号，在 a 的两侧分别为 * 运算符和 () 运算符。() 优先级高于 *，因此 a 先与 () 结合。这是函数形式。函数前面有一个 *，表示此函数是指针型函数(函数返回值是指针)。最前面的 int 表示返回的指针指向整型变量。

【例 7.14】 有若干个学生的成绩(每个学生有 4 门课程)，要求在用户输入学生序号以后，能输出该学生的全部成绩。用指针函数来实现。

【程序】

```
#include <stdio.h>
void main()
{
  float * search(float (*pointer)[4], int n);
  float score[][4]={{60, 70, 80, 90}, {56, 89, 67, 88}, {34, 78, 90, 66}};
  float *p;
  int i,m;
  printf("Enter the number of student: ");
  scanf("%d", &m);
  printf("The scores of No.%d are:\n", m);
  p= search(score,m);
  for (i=0; i< 4; i++)
      printf("%5.2f\t", (p+ i));
  printf("\n");
}
float * search(float (*pointer)[4], int n)
{
  float *pt;
  pt= *(pointer+n);
  return (pt);
}
```

【分析】 注意：学生序号是从 0 号算起的。函数 search 被定义为指针型函数，它的形参 pointer 是指向包含 4 个元素的一维数组的指针变量。pointer+1 指向 score 数组第 1 行。见图 7.28。*(pointer+1) 指向第 1 行第 0 列元素。pt 是指针变量，它指向实型变量(而不是指向一维数组)。main 函数调用 search 函数，将 score 数组的首地址传给 pointer(score 是指向行的指针，而不是指向列元素的指针)。m 是要查找的学生序号。调用 search 函数后，得到一个地址(指向第 m 个学生第 0 门课程)，赋给 p。然后将此学生的 4 门课的成绩打印出来。*(p+i) 表示此学生第 i 门课的成绩。

注意指针变量 p，pt 和 pointer 的区别。如果将 search 函数中的语句

更多高清PDF图书 www.gqpdf.com

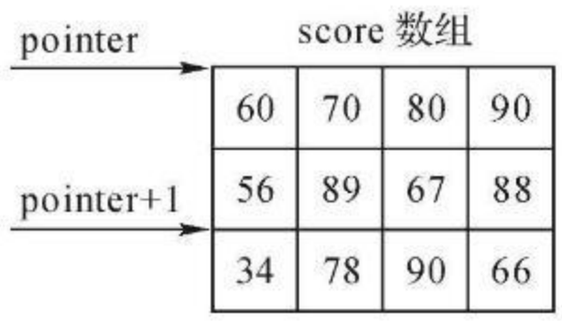


图 7.28　pointer+1 指向 score 数组第 1 行

```
pt=*(pointer+n);
```

改为：

```
pt=(*pointer+n);
```

得到的不是第一个学生的成绩，而是二维数组中 a[0][1] 开始的 4 个元素的值。

【运行结果】

```
Enter the number of student:1↙
The scores of No.%d are:
70.00  80.00  90.00  56.00
```

这是为什么呢？请读者思考。

【例 7.15】 对上例中的学生，找出其中有不及格课程的学生及其学生号。

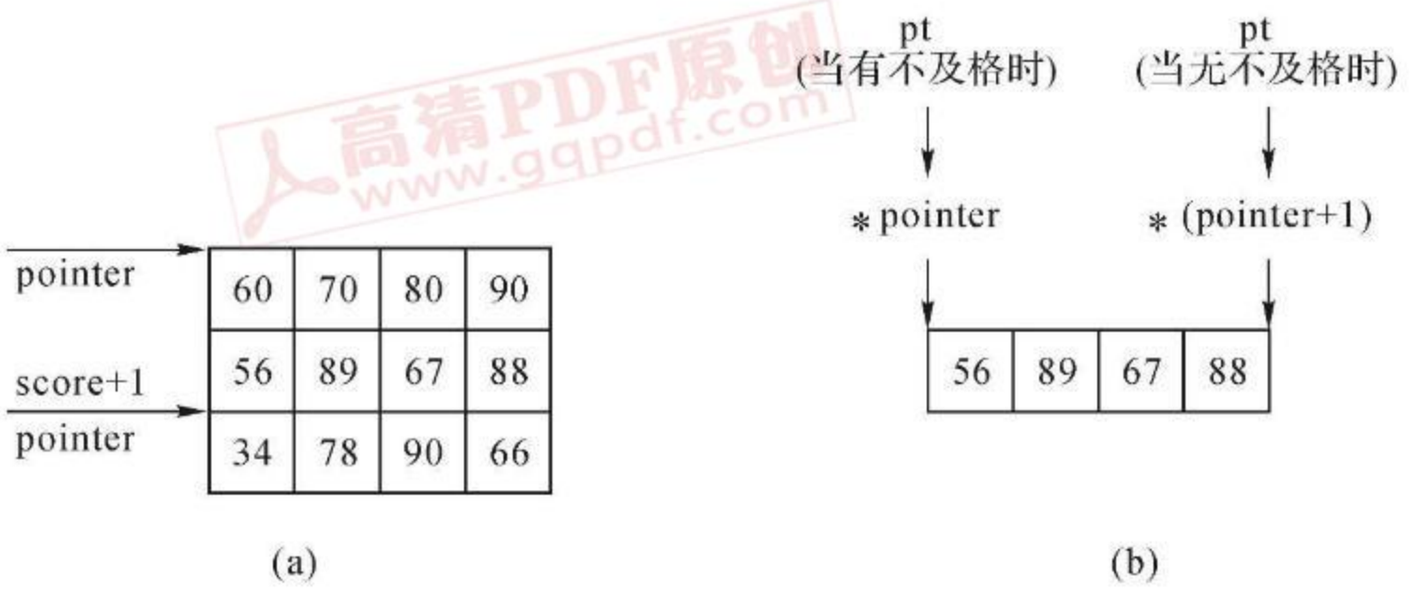
【程序】

```
#include<stdio.h>
void main()
{
  float *search(float(*pointer)[4]);
  float score[][4]={{60,70,80,90},{56,89,67,88},{34,78,90,66}};
  float *p;
  int i,j;
  for(i=0;i<3;i++)
  {
      p=search(score+i);
      if(p==*(score+i))
      {
          printf("No.%d scores:",i);
          for(j=0;j<4;j++)
              printf("%5.2f",*(p+j));
          printf("\n");
      }
  }
}
float *search(float(*pointer)[4])
{
  int i;
```

高精PDF原创 www.gqpdf.com

```
float *pt;
pt=*(pointer+1);
for(i=0; i<4; i++)
    if(*(*pointer+i)<60)
    {
        pt=*pointer;
        break;
    }
  return(pt);
}
```

【分析】 函数 search 的作用是检查一个学生有无不及格的课程。在 search 函数中 pointer 是指针变量,指向一维数组(有 4 个元素)。pt 为指向实型变量的指针变量。从实参传给形参 pointer 的是 score+i,它是 score 第 i 行的首地址,见图 7.29(a)。



(a) (b)

图 7.29

在 search 函数中,先使 pt=*(pointer+1),即把 pt 指向本行之末尾,见图 7.29(b)。这是为了区分有无不及格课程的标志。若经检查 4 门课中有不及格的,就使 pt 指向本行第 0 列元素。若无不及格则保持 pt 指向本行末尾(亦即下一行第 0 列元素)。将 pt 返回 main 函数。在 main 函数中,把调用 search 得到的函数值(指针变量 pt 的值)赋给 p。用 if 语句判断 p 是否等于 *(score+i),若相等,表示所查的学生有不及格课程(p 的值为 *(score+i),即 p 指向第 i 行的第 0 列元素)。若无不及格,p 的值是 *(score+i+1),因为在函数 search 中已使它指向本行的末尾,也就是下一行开头。如果 p 等于 *(score+i),就输出该学生(有不及格课程的学生)4 门课成绩。

7.5 指针数组和指向指针的指针

7.5.1 指针数组的概念

一个数组的元素均为指针类型的数据,称为指针数组。

更多高清PDF图书 www.gqpdf.com

定义一维指针数组的一般形式为：

类型名 * 数组名[数组长度]

例如：

```
int *p[4]
```

由于[]比 * 优先级高，因此 p 先与[4]结合，形成 p[4]形式，这是数组形式，它有4个元素。然后再与 p 前面的“*”结合，“*”表示此数组是指针类型的，每个数组元素(相当于一个指针变量)都可指向一个整型数据。

指针数组适合于用来指向若干个字符串，使字符串处理更加方便灵活。



活动9　对图书馆的藏书进行排序和查询。

例如，图书馆有若干本书，想把书名放在一个数组中(图7.30(a))，然后要对这些书目进行排序和查询。按一般方法，字符串本身就是一个字符数组。因此要设计一个二维的字符数组才能存放多个字符串。但在定义二维数组时，需要指定列数，也就是说二维数组中每一行中包含的元素个数(即列数)相等。而实际上各字符串(书名)长度一般是不相等的。如按最长的字符串来定义列数，则会浪费存储空间。见图7.30(b)。

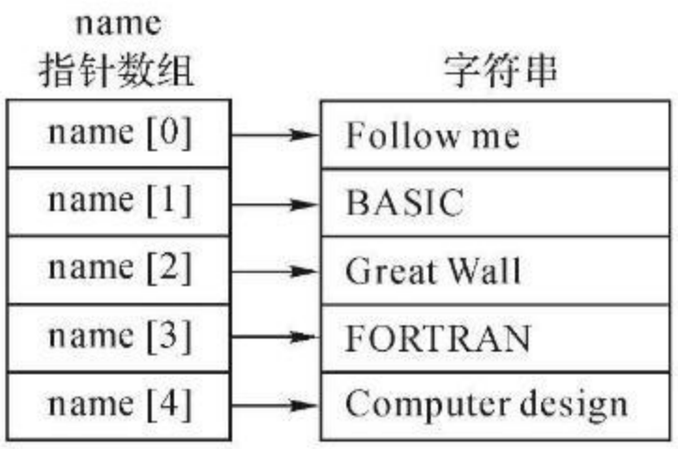
字符串

Follow me
BASIC
Great Wall
FORTRAN
Computer design

(a)

F	o	l	l	o	w		m	e	\0						
B	A	S	I	C	\0										
G	r	e	a	t		W	a	l	l	\0					
F	O	R	T	R	A	N	\0								
C	o	m	p	u	t	e	r		d	e	s	i	g	n	\0

(b)



(c)

图7.30　指针数组 name

可以分别定义一些字符串，然后用指针数组中的元素分别指向各字符串，见图7.30(c)。如果想对字符串排序，不必改动字符串的位置，只需改动指针数组中各元素的指向(即改变各元素的值，这些值是各字符串的首地址)。这样，各字符串的长度可以不同，而且移动指针变量的值(地址)要比移动字符串所花的时间少得多。

【例7.16】 将若干字符串按字母顺序(由小到大)输出。

【程序】

```
#include<stdio.h>
```

```
#include<string.h>
void main()
{
  void sort(char *name[], int n);
  void print(char *name[], int n);
  char *name[]={"Follow me","BASIC","GreatWall","FORTRAN","ComputerDesign"};
  int n=5;
  sort(name, n);
  print(name, n);
}
void sort(char *name[], int n)
{
  char *temp;
  int i, j, k;
  for(i=0; i<n-1; i++)
  {
      k=i;
      for(j=i+1; j<n; j++)
          if(strcmp(name[k], name[j])>0)
              k=j;
      if(k!=i)
      {
          temp=name[i];
          name[i]=name[k];
          name[k]=temp;
      }
  }
}
void print(char *name[], int n)
{
  int i;
  for(i=0; i<n; i++)
      printf("%s\n", name[i]);
}
```

高清PDF原创 www.gqpdf.com

【分析】 在 main 函数中定义指针数组 name。它有 5 个元素,其初值分别是“Follow me”、“BASIC”、“Great Wall”、“FORTRAN”,“Computer Design”的首地址。见图 7.30(C)。这些字符串是不等长的。sort 函数的作用是对字符串排序。sort 函数的形参 name 也是指针数组名,接受实参传过来的 name 数组的首地址,因此形参 name 数组和实参 name 数组指的是同一数组。用选择法对字符串排序。strcmp 是字符串比较函数,name[k]和 name[j]是第 k 个和第 j 个字符串的起始地址。strcmp(name[k], name[j])的值为:如果 name[k]所指的字符串大于 name[j]所指的字符串,则此函数值为正值;若相等,则函数

更多高清PDF图书 www.gqpdf.com

值为0;若小于,则函数值为负值。if 语句的作用是将两个串中“小”的那个串的序号(i 或 j 之一)保留在变量 k 中。当执行完内循环 for 语句后,从第 i 个串到第 n 个串这么多字符串中,第 k 个串最“小”。若 k ≠ i 就表示最小的串不是第 i 串。故将 name[i] 和 name[k] 对换,也就是将指向第 i 个串的数组元素(是指针型元素)与指向第 k 个串的数组元素对换。执行完 sort 函数后指针数组的情况如图 7.31 所示。

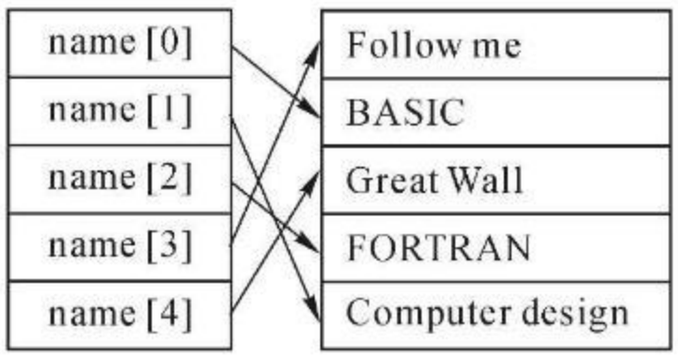


图 7.31　指针数组 name 交换后

print 函数的作用是输出各字符串。name[0] 到 name[4] 分别是各字符串(按从小到大顺序排好序的各字符串)的首地址(按字符串从小到大顺序,name[0] 指向最小的串),用 "%s" 格式符输出,就得到这些字符串。

print 函数也可改写为以下形式:

```
void print(char *name[], int n)
{
   int i=0;
   char p;
   p=name[0];
   while(i<n)
   {
       p=*(name+i++);
       printf("%s\n",p);
   }
}
```

其中“*(name+i++);”表示先求 *(name+i) 的值,即 name[i](它是一个地址),然后使 i 加 1。在输出时,按字符串形式输出以 p 地址开始的字符串。

高清PDF原创 www.gqpdf.com

7.5.2　指向指针的指针

指向指针型数据的指针变量简称为指向指针的指针。从图 7.32 可以看到,name 是一个指针数组,它的每一个元素是一个指针型数据,其值为地址。name 是一个数组,它的每一元素都有相应的地址。数组名 name 代表该指针数组的首地址。name+i 是 name[i] 的地址。name+i 就是指向指针型数据的指针(地址)。可以设置一个指针变量 p,它指向指针数组的元素。p 就是指向指针型数据的指针变量。

定义指向指针型数据的指针变量的一般形式:

类型名 * * 指向指针型数据的指针变量名

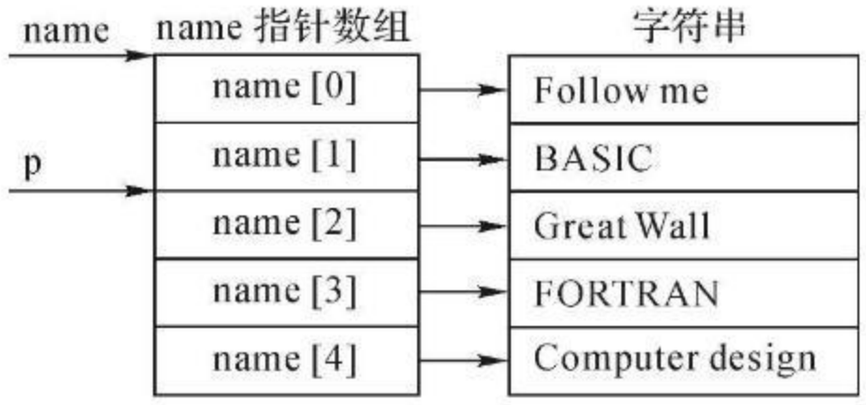


图 7.32　指向指针的指针

例如：

```
char * * p;
```

p 的前面有两个 * 号。* 运算符的结合性是从右到左，因此 * * p 相当于 *(*p)，*p 是指针变量的定义形式，它前面又有一个 * 号，表示指针变量 p 是指向一个字符指针变量(即指向字符型数据的指针变量)的。*p 就是 p 所指向的另一个指针变量，如果有：

```
p=name+2;
printf("%o\n", *p);
printf("%s\n", *p);
```

第一个 printf 函数语句输出 name[2] 的值(是一个地址)，第二个 printf 函数语句以字符串形式(%s)输出字符串“Great Wall”。

【例 7.17】 使用指向指针的指针。

【程序】

```
#include<stdio.h>
voidmain()
{
  char *name[]={"Followme", "BASIC", "GreatWall", "FORTRAN", "ComputerDesign"};
  char * *p;
  int i;
  for(i=0; i<5; i++)
  {
      p=name+i;
      printf("%s\n", *p);
  }
}
```

【分析】 p 是指向指针的指针变量，在第一次执行循环体时，赋值语句“p=name+i;”使 p 指向 name 数组的 0 号元素 name[0]，*p 是 name[0] 的值，即第一个字符串的起始地址，用 printf 函数输出第一个字符串(格式符为 %s)。依次输出 5 个字符串。

指针数组的元素也可以不指向字符串，而指向整型数据或实型数据等，例如：

```
int a[5]={1, 3, 5, 7, 9, i};
int *num[5];
int * *p;
```

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

```
for(i=0; i< 5; i++)
num[i]=&a[i];
```

此时为了得到 a[2] 中的数据“5”,可以先使 p=num+2,然后输出 **p。注意 *p 是 p 间接指向的对象的地址 num[2]。而 **p 是 p 间接指向的对象的值,即 *num[2],也就是 a[2] 的值 5。见图 7.33。

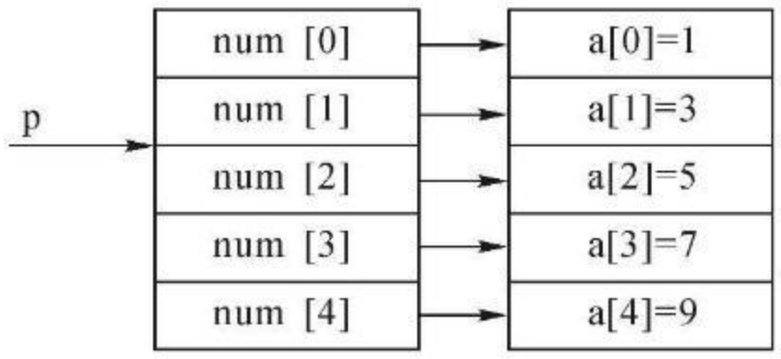


图 7.33　指向整型数据的指针数组

【例 7.18】 指向整型数据的指针数组。

【程序】

```
#include<stdio.h>
voidmain()
{
   static int a[5]={1, 3, 5, 7, 9};
   int *num[5]={&a[0],&a[1],&a[2],&a[3],&a[4]};
   int **p, i;
   p=num;
   for(i=0; i< 5; i++)
   {
       printf("%d\t", **p);
       p++;
   }
   printf("\n");
}
```

【分析】 利用指针变量访问另一个变量就是“间接访问”。如果在一个指针变量中存放一个目标变量的地址,这就是“单级间址”,指向指针的指针用的是“二级间址”方法。从理论上说,间址方法可以延伸到更多的级。但实际上在程序中很少有超过二级间址的。级数愈多,愈难理解,容易产生混乱,出错机会也多。

7.6　本章小结

7.6.1　指针数据类型的小结

一般数据类型与指针数据类型的定义和含义见表 7.3。

高清PDF原创 www.gqpdf.com

表 7.3 一般数据类型与指针数据类型的定义和含义

定 义	含 义
int i;	定义整型变量 i
int *p;	定义 p 为指向整型数据的指针变量
int a[n];	定义整型数组 a,它有 n 个整型元素
int *p[n];	定义指针数组 p,它有 n 个指向整型数据的指针元素
int (*p)[n];	定义 p 为指向有 n 个整型元素的一维数组的指针变量
int f();	定义 f 为返回整型函数值的函数
int *p();	定义 p 为返回指针的函数,该指针指向整型数据
int (*p)();	定义 p 为指向函数的指针,该函数返回整型函数值
int **p;	定义 p 为指针变量,它指向一个指向整型数据的指针变量

7.6.2 指针运算小结

(1)指针变量加(减)一个整数

例如:p++,p--,p+i,p-i,p+=i,p-=i 等。

C 语言规定:一个指针变量加(减)一个整数并不是简单地将指针变量的原值加(减)一个整数,而是将该指针变量的原值(是一个地址)和它指向的变量所占用的内存单元字节数相加(减)。

如:p+i 代表地址计算:p+c * i。c 是它指向的变量所占用的内存单元字节数,这样才能保证 p+i 指向 p 下面的第 i 个元素,它才有实际意义。

(2)指针变量赋值

将一个变量地址赋给一个指针变量。如表 7.4:

表 7.4 将一个变量地址赋给一个指针变量

p=&a	(将变量 a 的地址赋给 p)
p=array;	(将数组 array 首地址赋给 p)
p=&array[i];	(将数组 array 第 i 个元素的地址赋给 p)
p=max;	(max 为已定义的函数,将 max 的入口地址赋给 p)
p1=p2;	(p1 和 p2 都是指针变量,将 p2 的值赋给 p1)

注意:不应把一个整数赋给指针变量。如:

```
p=1000;
```

只能将变量已分配的地址赋给指针变量。同样也不应把指针变量 p 的值(地址)赋给一个整型变量 i:

```
i=p;
```

更多高清PDF图书 www.gqpdf.com

(3)指针变量可以有空值,即该指针变量不指向任何变量,如:

```
p=NULL;
```

实际上 NULL 是整数0,它使 p 的存储单元中所有二进位均为0,也就是使 p 指向地址为0的单元。系统保证使该单元不作它用(不存放有效数据),即有效数据的指针不指向0单元。实际上是先定义 NULL,即

```
#defineNULL 0
...
p=NULL;
```

高清PDF原创 www.gaopdf.com

在 studio.h 头文件中就有以上的 NULL 定义,它是一个符号常量。用"p=NULL;"表示 p 不指向任一有用单元。

注意:p 的值为 NULL 与未对 p 赋值是两个不同的概念。前者是有值的(值为0),不指向任何变量,后者虽未对 p 赋值但并不等于 p 无值,只是它的值是一个无法预料的值,也就是 p 可能指向一个事先未指定的单元。这种情况是很危险的。因此,在引用指针变量之前应对它赋值。

任何指针变量或地址都可以与 NULL 作相等或不相等的比较,如

```
if(p == NULL)
.....
```

(4)如果两个指针变量指向同一个数组的元素,则两个指针变量可以相减,差是两个指针变量之间的元素个数(图7.34)。p1 指向 a[1],p2 指向 a[4],p2-p1=4-1=3。

但两个指针变量相加无实际意义。

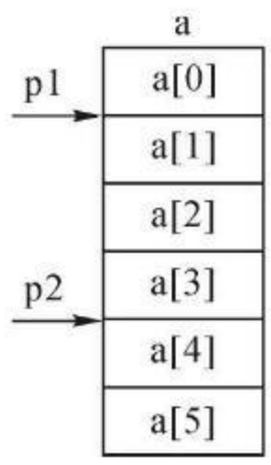


图7.34　两个指针变量值之差是两个指针之间的元素个数

(5)如果两个指针变量指向同一个数组的元素,则可以进行比较。指向前面的元素的指针变量"小于"指向后面元素的指针变量。如图7.34 中,p1 < p2,或者说,表达式"p1 < p2"的值为1(真),而"p2 < p1"的值为0(假)。

注意:如果 p1 和 p2 不指向同一数组则比较无意义。

课后练习

1. 输入3个整数,按由小到大的顺序输出。

2. 输入3个字符串,按由小到大的顺序输出。

3. 输入 10 个整数,将其中最小的数与第一个数对换,把最大的数与最后一个数对换。写 3 个函数:(1)输入 10 个数;(2)进行处理;(3)输出 10 个数。

4. 写一个函数,将一个 3×3 的矩阵转置。

5. 有一个班 4 个学生,5 门课。(1)求第一门课的平均分;(2)找出有 2 门以上课程不及格的学生,输出他们的学号和全部课程成绩和平均成绩;(3)找出平均成绩在 90 分以上或全部课程成绩在 85 分以上的学生。分别编 3 个函数实现以上 3 个要求。

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

高清PDF原创 www.gqpdf.com

第 8 章

结构体

【本章主要内容】

本章主要介绍C语言的另外3种结构数据类型:结构体、公用体和枚举类型,以及用结构体建立和处理链式存储结构——链表等。

【本章要求】

(1)掌握结构体类型数据的定义和成员的引用。
(2)掌握用 typedef 说明一个新类型。
(3)掌握通过结构体构成链表,单向链表的建立、结点数据的输出、删除与插入。
(4)了解共用体类型数据的定义和成员的引用。
(5)了解枚举数据类型的定义和引用。

8.1 结构体

8.1.1 结构体的定义

迄今为止,我们已经学习了基本数据类型(如整型、实型、字符型等),也学习了一种结构化数据类型——数组。数组中的各元素是属于同一种基本类型的。但是只有这些数据类型是不够的。有时需要将不同类型的数据组合成一个有机的整体,以便于引用。这些组合在一个整体中的数据是互相联系的。

活动1 基本数据类型的局限。

在学籍管理系统中,每个学生的记录包括学号、姓名、性别、年龄、成绩、家庭地址等字段。这些字段都与某一个具体学生相联系,如图8.1所示。我们可以看到性别(sex)“M”、

年龄(age)“18”、成绩(score)“87 .5”、地址(addr)“Beijing”是属于学号为“10010”和名为“Li Fun”的学生的。如果将 num , name , sex , age , score , addr 分别定义为互相独立的基本数据类型的变量,则难以反映它们之间的内在联系。

num	name	sex	age	score	addr
10010	LiFun	M	18	87 .5	Beijing

图 8 .1 学生数据项

如果把它们当做一个整体来对待,即组织成一个组合项,在一个组合项中包含若干个类型相同或不同的数据项,则能够充分反映出它们之间的内在联系。在 C 语言中,允许用户自己定义这样的数据结构。这种数据结构称为结构体(Structure)。例如我们可以将每个学生的记录定义为一种结构体:

```
struct student
{
int num;
char name[20];
char sex;
int age;
float score;
char name[40];
};
```

注意:不要忽略右花括号右边的分号。

这里,我们定义了一种新的结构体类型 struct student(struct 是定义结构体类型时所必须使用的关键字,不能省略),它包括 num , name , sex , age , score , addr 不同类型的数据项。struct student 是一个类型名,它和系统提供的标准类型(如 int , char , float , double 等)一样具有同样的地位和作用,都可以用来定义变量的类型。

由上例我们可以得出声明一个结构体类型的一般形式为

struct 结构体名

{

/* **成员表列** */

类型名 成员名

};

花括号内是该结构体中的各个成员,由它们组成一个结构体。对各成员都应进行类型声明。“成员表列”又称为“域表”。每一个成员也称为结构体中的一个域。结构体名和成员名的命名遵循标识符命名规则。

8 .1 .2 结构体变量的定义、初始化和引用

1 .结构体变量的定义

定义一个结构体类型相当于定义一个模型,其中并无具体数据,系统对之也不分配内存

更多高清PDF图书 www.gqpdf.com

单元。应当在定义结构体类型后定义该结构体类型的变量,并在其中存放具体数据,在程序中才能使用该结构体类型。

定义结构体变量有三种方法:

(1)先声明结构体类型再定义变量名

一般形式为:

结构体类型名 结构体变量名;

如:

struct student student1, student2;

定义了 student1 和 student2 为 struct student 类型的变量,它们具有 struct student 类型的结构。如图 8.2 所示。

10001	Zhang Xin	M	19	90.5	Shanghai
10002	Wang Li	F	20	98	Beijing

图 8.2　struct student 类型变量

在定义了结构体变量后,系统会为之分配内存单元。

(2)在声明结构体类型的同时定义变量

一般形式为:

struct 结构体名

{

/* 成员表列 */

类型名 成员名

}变量名表列;

例如:

```
struct student
{
int num;
char name[20];
char sex;
int age;
float score;
char addr[40];
}student1, student2;
```

(3)直接定义结构类型变量,这种形式没有结构体名

一般形式为:

struct

{

/* 成员表列 */

类型名 成员名

}变量名表列;

高清PDF原创 www.gqpdf.com

结构体类型的几点说明：

(1)类型与变量是不同的概念：只能对变量赋值、存取或运算，而不能对一个类型赋值、存取或运算；在编译时，对类型不分配空间，对变量分配空间。

(2)成员也可以是一个结构体变量。

如：

```
struct date      /* 声明一个结构体类型 */
{
int month;
int day;
int year;
};

struct student
{
int num;
char name[20];
char sex;
int age;
struct date birthday;     /* birthday 是 struct date 类型 */
char name[40];
}student1, student2;
```

这里先声明一个 struct date 类型代表“日期”，包括 3 个成员：month(月)、day(日)、year(年)。再声明 struct student 类型时，将成员 birthday 定义为 struct date 类型(见图 8.3)

num	name	sex	age	birthday			addr
				month	day	year	

图 8.3 struct student 的结构

(3)结构体的成员名可以与程序中的变量名相同，但二者是不同的对象。

2.结构体变量的初始化

结构体变量可以在定义时指定其各个成员的初值。

【例 8.1】 对结构体变量初始化。

【程序】

```
#include <stdio.h>

void main()
{
struct student
{
   long int num;
```

更多高清PDF图书 www.gqpdf.com

```
    char name[20];
    char sex;
    char name[40];
} a={89031, "Li Lin", 'M', "123 Beijing Road"};

printf("NO.:%ld\nname:%s\nsex:%c\naddress:%s\n", a.num, a.name, a.sex, a.addr);
}
```

3.结构体变量的引用

结构体变量的引用规则：

(1)结构体变量中的各个成员只能分别进行输入和输出。引用结构体变量中成员的方式为：

结构体变量名.成员名

例如：student1.num 表示 student1 变量中的 num 成员，即 student1 的 num(学号)项。

可以对变量的成员赋值，例如：

```
student1.num=10010;
```

“.”是成员(分量)运算符，在所有的运算符中优先级最高，因此可以把 student1.num 作为一个整体来看待。上面赋值语句的作用是将整数 10010 赋给 student1 变量中的成员 num。

注意：不能将一个结构体变量作为一个整体进行输入和输出。例如，已定义 student1 为结构体变量并且它们已有值。不能这样引用：

```
printf("%d,%s,%c,%d,%f,%s\n", student1);
```

(2)如果成员本身又属一个结构体类型，则要用若干个成员运算符，一级一级地找到最低一级的成员。只能对最低一级的成员进行赋值或存取以及运算。例如：

```
student1.birthday.month
```

(3)对结构体变量的成员可以像普通变量一样根据其类型进行相应的各种运算。

(4)既可以引用结构体变量成员的地址，也可以引用结构体变量的地址。如：

```
printf("%o",&student1.num);      /*输出 student1.num 的地址*/
printf("%o",&student1);          /*输出 student1 的首地址*/
```

但不能用以下语句整体读入结构体变量，如：

```
scanf("%d,%s,%c,%d,%f,%s",&student1);
```

结构体变量的地址主要用于作函数参数，传递结构体的地址。

8.2　结构体数组

结构体数组即数组中的每个数组元素都是一个结构体类型的数据，它们都分别包括各

个成员(分量)项。

8.2.1 结构体数组的定义

结构体数组的定义和结构体变量的定义的方法相仿,只需说明其为数组即可。如:

```
struct student
{
int num;
char name[20];
char sex;
int age;
float score;
char name[40];
};
struct student stu[3];
```

以上定义了一个数组 stu,其元素为 struct student 类型的数据,数组有 3 个元素。

结构体数组的各元素在内存中连续存放,见图 8.4。



图 8.4 结构体数组元素在内存中连续存放

8.2.2 结构体数组的初始化和应用

1.结构体数组的初始化

结构体数组初始化的一般形式是在定义数组的后面加上:={初值表列};如:

```
struct student
{
int num;
char name[20];
char sex;
int age;
float score;
char name[40];
} stu[3]={{10101,"LiLin",'M',18,87.5,"103 Beijing Road"},{10102,"Zhang Fun",'M',19,
99,"130 ShanghaiRoad"},{10104,"WangMin",'F',20,78.5,"1010 Zhongshan Road"}};
```

与普通类型的数组类似,在定义结构体数组时,元素个数可以不指定,编译时,系统会根

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

据给出初值的结构体常量的个数来确定数组元素的个数。

2.结构体数组应用举例

【例8.2】 对候选人得票统计程序。设有3个候选人,每次输入一个得票的候选人的名字,要求最后输出各人得票结果。

【程序】

```
#include<stdio.h>
#include<string.h>

struct person
{
char name[20];
int count;
} leader[3]={"Li", 0, "Zhang", 0, "Fun", 0};

void main()
{
int i, j;
char leader_name[20];

for(i=1; i<=10; i++)
{
  scanf("%s", leader_name);

  for(j=0; j<3; j++)
      if(strcmp(leader_name, leader[j].name) == 0)
          leader[j].count++;
}

printf("\n");

for(i=0; i<3; i++)
  printf("%5s:%d\n", leader[i].name, leader[i].count);
}
```

高清PDF原创 www.gqpdf.com

【分析】 程序定义一个全局的结构体数组leader,它有3个元素,每一元素包含两个成员name(姓名)和count(票数),在定义数组时使之初始化,将3位候选人的票数都先置零。见图8.6。

在主函数中定义字符数组leader_name,它代表被选人的姓名,在10次循环中每次先输入一个被选人的具体人名,然后把它与3个候选人姓名相比,看它和哪一个候选人的名字相同。注意:leader[j]是数组leader的第j个元素,它包含两个成员项,leader_name应该和leader数组第j个元素的name成员相比。

name	count
Li	0
Zhang	0
Fun	0

图 8.6 结构体数组 leader

若 j 为某一值时,输入的姓名与 leader[j].name 相等,就执行“leader[j].count++”。成员运算符“.”优先于自增运算符“++”,因此它相当于(leader[j].count)++,使 leader[j]的成员 count 的值加 1。在输入和统计结束之后,将 3 人的名字和得票数输出。

8.3 结构体类型数据的指针

一个结构体变量的指针是该结构体变量所占据的内存单元的起始地址。指针变量不仅可以用来指向一个结构体变量(此时该指针变量的值是结构体变量的起始地址),也可以用来指向结构体数组中的元素。

8.3.1 结构体变量的指针

【例 8.3】 指向结构体变量的指针的应用。

【程序】

```
#include<stdio.h>
#include<string.h>

voidmain()
{
struct student
{
   long num;
   char name[20];
   char sex;
   float score;
};

struct student stu_1;
struct student *p;

p=&stu_1;
```

更多高清PDF图书 www.gqpdf.com

```
    stu_1.num=89101;
    strcpy(stu_1.name,"LiLin");
    stu_1.sex='M';
    stu_1.score=89.5;
    printf("No.:%ld\nname:%s\nsex:%c\nscore:%f\n",stu_1.num,stu_1.name,stu_1.sex,
    stu_1.score);
    printf("No.:%ld\nname:%s\nsex:%c\nscore:%f\n",(*p).num,(*p).name,(*p).sex,
    (*p).score);
}
```

【分析】 在主函数中声明了 struct student 类型,然后定义一个 struct student 型的变量 stu_1。同时又定义一个指针变量 p,指向一个 struct student 类型的数据。在函数的执行部分将结构体变量 stu_1 的起始地址赋给指针变量 p,使 p 指向 stu_1(见图 8.7),然后对 stu_1 的各成员赋值。printf 函数是输出 stu_1 的各个成员的值。stu_1.num 表示 stu_1 中的成员 num,其余类推。第二个 printf 函数也是用来输出 stu_1 各成员的值,使用的是(*p).num 这样的形式。(*p)表示 p 指向的结构体变量,(*p).num 是 p 指向的结构体变量中的成员 num。

注意:*p 两侧的括号不能省略,因为“.”优先于“*”运算符,*p.num 等价于*(p.num)。

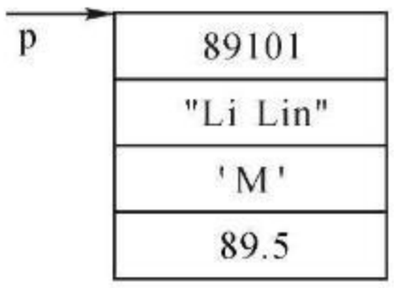


图 8.7　指针变量 p 指向结构体变量

两个 printf 函数输出的结果是相同的。

在 C 语言中,以下三种形式等价:

①结构体变量名.成员名。

②(*结构体变量的指针名).成员名。

③结构体变量的指针名->成员名。

“->”是指向运算符。

8.3.2　结构体数组的指针

【例 8.4】 指向结构体数组的指针的应用。

【程序】

```
#include<stdio.h>

struct student
{
```

```
    int num;
    char name[20];
    char sex;
    int age;
    };

    struct student stu[3]={{10101, "LiLin", 'M', 18}, {10102, "Zhang Fun", 'M', 19}, {10104, "
WangMin", 'F', 20}};

    void main()
    {
    struct student *p;

    printf("No. Name                    sex age\n");

    for (p=stu; p< stu+3; p++)
      printf("%5d%-20s%2c%4d\n", p->num, p->name, p->sex, p->age);
    }
```

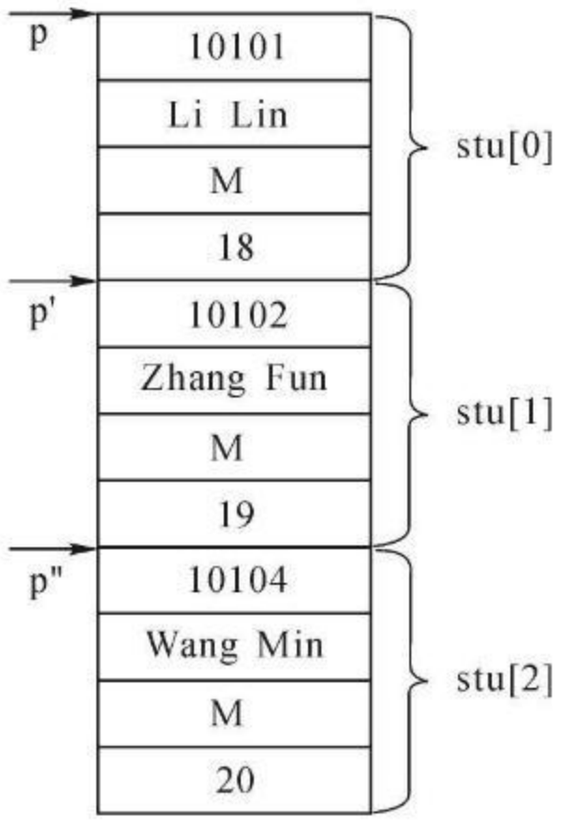


图 8.8　p 为指向结构体的指针

p 是指向 struct student 结构体类型数据的指针变量。在 for 循环语句中先使 p 的初值为 stu(数组 stu 的起始地址,见图 8.8 中 p 的指向)。在第一次循环中输出 stu[0]的各个成员值。然后执行 p++,使 p 自加 1。p 加 1 意味着 p 所增加的值为结构体数组 stu 的一个元素所占的字节数。执行 p++后,p 指向 stu[1]的起始地址,见图 8.8 中 p' 的指向。以下依此类推。

8.3.3　结构体类型数据的指针作函数参数

将一个结构体变量的值传递给另一个函数,有 3 种方法:

更多高清PDF图书　www.gqpdf.com

(1)用结构体变量作实参。用结构体变量作实参时,采取的是“值传递”的方式,将结构体变量所占的内存单元的内容全部顺序传递给形参。形参也必须是同类型的结构体变量。在函数调用期间形参也要占用内存单元。一般较少用这种方法。

(2)用结构体变量的成员作参数。将实参值传给形参。用法和用普通变量作实参是一样的,属于“值传递”方式。应当注意实参与形参的类型保持一致。

(3)用指向结构体变量(或数组)的指针作实参,将结构体变量(或数组)的地址传给形参。

【例8.5】 有一个结构体变量stu,内含学生学号、姓名和3门课的成绩。要求在main函数中赋值,在另一函数print中将它们打印输出。用结构体变量作函数参数。

【程序】

```
#include<stdio.h>
#include<string.h>

#define FORMAT "%d\n%s\n%f\n%f\n%f\n"

struct student
{
int num;
char name[20];
float score[3];
};

void main()
{
void print(struct student);

struct student stu;

stu.num=12345;
strcpy(stu.name, "LiLi");
stu.score[0]=67.5;
stu.score[1]=89;
stu.score[2]=78.6;
print(stu);
}

void print(struct student stu)
{
printf(FORMAT, stu.num, stu.name, stu.score[0], stu.score[1], stu.score[2]);
printf("\n");
}
```

【分析】 main 函数中的 stu 定义为 struct studen 类型变量,print 函数中的形参 stu 也定义为 struct student 类型变量。在 main 函数中对 stu 的各成员赋值。在调用 print 函数时以 stu 为实参向形参 stu 实行“值传递”。在 print 函数中输出结构体变量 stu 各成员的值。

【例 8.6】 将例 8.5 改用指向结构体变量的指针作实参。

【程序】

```
#include<stdio.h>

#define FORMAT "%d\n%s\n%f\n%f\n%f\n"

struct student
{
int num;
char name[20];
float score[3];
} stu={12345,"LiLi",67.5,89,78.6};

void main()
{
void print(struct student *);     /*形参类型修改成指向结构体的指针变量*/

print(&stu);     /*实参改为 stu 的起始地址*/
}

void print(struct student *p)     /*形参类型修改了*/
{
printf(FORMAT,p->num,p->name,p->score[0],p->score[1],p->score[2]);
/*用指针变量调用各成员之值*/
printf("\n");
}
```

print 函数中的形参 p 被定义为指向 struct student 类型数据的指针变量。调用 print 函数时,用结构体变量 stu 的起始地址 &stu 作实参。在调用函数时将该地址传送给形参 p,使 p 指向 stu,见图 8.9。在 print 函数中输出 p 所指向的结构体变量的各个成员值,它们实际上就是 stu 的成员值。

8.4 用 typedef 定义类型

C 语言中,关键字 typedef 用于声明新的类型名来代替已有的类型名。如:

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

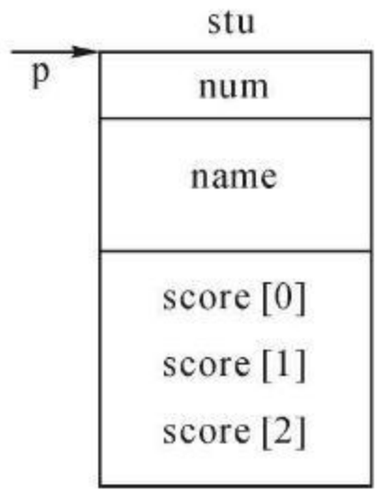


图 8.9　指向结构体变量的指针作实参

```
typedef int INTEGER;
typedef float REAL;
```

以上两行代码表示指定用 INTEGER 代表 int 类型,REAL 代表 float。这样,以下两行等价:

```
int i, j; float a, b;
INTEGER i, j; REAL a, b;
```

因此,声明一个类型新的类型名的方法是:

①先按定义变量的形式书写;

②将变量名换成新类型名;

③在前面加 typedef;

④用新类型名定义变量。

对于数组:

①先按定义变量的形式书写:int n[100];

②将变量名换成新类型名:int NUM[100];

③在前面加 typedef:ypedef int NUM[100];

④用新类型名定义变量:NUM n。

对于指针:

① char *p;

② char *STRING;

③ typedef char *STRING;

④ STRING p, s[10]。

一般情况下,我们习惯上常把用 typedef 声明的类型名用大写字母表示,以便与系统提供的标准类型标识符相区别。

说明:

(1)用 typedef 可以声明各种类型名,但不能用来定义变量。

(2)用 typedef 只是对已经存在的类型增加一个类型名,而没有创造新的类型。

(3)typedef 与#define 有相似之处,如:

typedef int COUNT;

和

#define COUNT int

的作用都是用 COUNT 代表 int。但事实上,它们二者是不同的。#define 是在预编译时处理的,它只能作简单的字符串替换,而 typedef 是在编译时处理的,它并不是作简单的字符串替换,而是采用如同定义变量的方法那样来声明一个类型。

(4)当不同的源文件中用到同一类型数据(尤其是像数组、指针、结构体、共用体等类型数据)时,常用 typedef 声明一些数据类型名,把它们单独放在一个文件中,然后在需要用到它们的文件中用#include 命令把它们包含进来。

(5)使用 typedef 有利于程序的通用与移植。例如,有的计算机系统 int 型数据用两个字节,数值范围为 -32768~32767,而另外一些机器则以 4 个字节存放一个整数,数值范围为±21 亿。如果把一个 C 程序从一个以 4 个字节存放整数的计算机系统移植到以 2 个字节存放整数的系统,按一般办法需要将定义变量中的每个 int 改为 long。如果程序中有多处用 int 定义变量,则要改动多处。但如果预先使用 INTEGER 来声明 int:

```
typedef int INTEGER;
```

那么,在程序中所有整型变量都用 INTEGER 定义,在移植时只需改动 typedef 定义体即可:

```
typedef long INTEGER;
```

8.5 用结构体处理链表

8.5.1 链表概述

链表是一种常见的重要的数据结构,是一种动态地进行存储分配的结构。用数组存放数据时,必须事先定义固定的长度(即元素个数)。比如,有的班级有 100 人,而有的班级只有 30 人,如果要用同一个数组先后存放不同班级的学生数据,则必须定义长度为 100 的数组。如果事先难以确定一个班的最多人数,则必须把数组定得足够大,以便能存放任何班级的学生数据,但是这会浪费内存。而链表则没有这种缺点,它是根据需要开辟内存单元。图 8.10 所示是最简单的一种链表(单向链表)的结构。

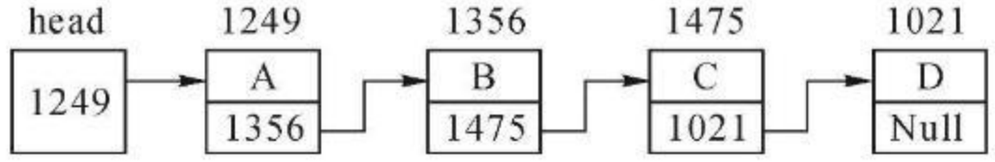


图 8.10 单向链表

链表有一个“头指针”变量,存放一个地址(图中以 head 表示),该地址指向一个元素。链表中的每一个元素称为“结点”,每个结点都应包括两个部分:一是用户的数据(数据部分);二为下一个结点的地址(地址部分)。head 指向第一个元素,第一个元素又指向第二个元素……直到最后一个元素。该元素不再指向其他元素,称为“尾节点”,地址部分存放

更多高清PDF图书 www.gqpdf.com

“NULL”(“空地址”,表示不指向任何结点),链表到此结束。

链表中各结点在内存中可以是不连续存放的。如果要找某一结点,必须先找到上一个结点,根据它提供的下一结点地址才能找到下一个结点。如果不提供“头指针”,则整个链表都无法访问。链表如同一条铁链一样,一环扣一环,中间是不能断开的。

因为一个结点中应包含一个指针变量,用它存放下一结点的地址,所以链表的数据结构必须利用指针变量才能实现。可以用结构体类型作链表中的结点。一个结构体类型包含若干成员,这些成员可以是数值类型、字符类型、数组类型,但至少要有一个是指针类型,用它来存放下一个结点的地址。例如:

```
struct student
{
int num;
float score;
struct student next;
};
```

成员 num 和 score 用来存放结点中的数据部分,相当于图 8.10 结点中的 A,B,C,D。next 是指针类型的成员,指向 struct student 类型数据(即指针部分)。一个指针类型的成员既可以指向其他类型的结构体数据,也可以指向自己所在的结构体类型的数据。现在 next 是 struct student 类型中的一个成员,它又指向 struct student 类型的数据。用这种方法就可以建立链表。见图 8.11。

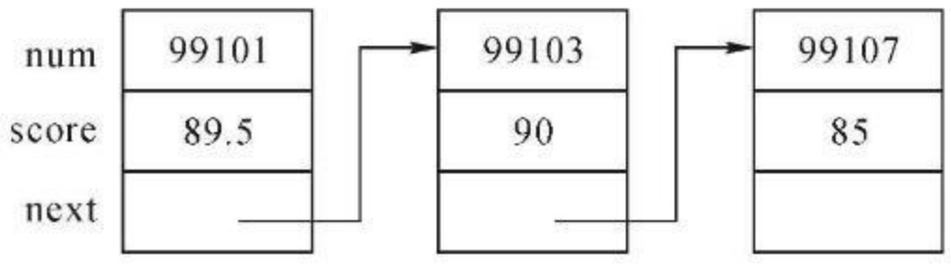


图 8.11　用指向自己所在的结构体类型的数据的指针变量建立链表

图中每一个结点都属于 struct student 类型,它的成员 next 存放下一结点的地址。

8.5.2　处理链表所需的函数

多数 C 语言的开发环境都包含了动态地开辟和释放存储单元的有关函数,在调用这些函数时必须包含 stdlib.h 头文件。

1.malloc 函数

函数原型为:

void * malloc(unsigned int size);

作用是在内存的动态存储区中分配一个长度为 size 的连续空间。此函数的返回值是一个指向分配区域起始地址的指针(基类型为 void);如果此函数未能成功地执行(例如内存空间不足),则返回空指针(NULL)。

高清PDF图书 www.gqpdf.com

2 .calloc 函数

函数原型为：

void * calloc(unsigned n , unsigned size);

作用是在内存的动态区存储中分配 n 个长度为 size 的连续空间。函数返回一个指向分配区域起始地址的指针；如果此函数未能成功地执行(例如内存空间不足)，则返回空指针(NULL)。

3 .free 函数

函数原型为：

void free(void * p);

作用是释放由指针变量 p 指向的内存区域，使这部分内存区域能被其他程序和变量使用。p 是调用 calloc 或 malloc 函数时返回的值。free 函数无返回值。

8 .5 .3 链表的建立

建立链表是指在程序执行过程中从无到有地建立起一个链表，即一个一个地开辟结点和输入各结点数据，并建立起前后相链的关系。

【例 8 .7】 编写一个函数，建立一个有 3 名学生数据的单向链表。

算法(见图 8 .12)：设 3 个指针变量：head，p1，p2，它们都是用来指向 struct student 类型数据的。先用 malloc 函数开辟第一个结点，并使 p1，p2 指向它。然后从键盘输入一个学生的数据给 p1 所指的第一个结点。程序中规定学号不能为零，如果输入的学号为 0，则表示建立链表的过程完成，该结点不应连接到链表中。

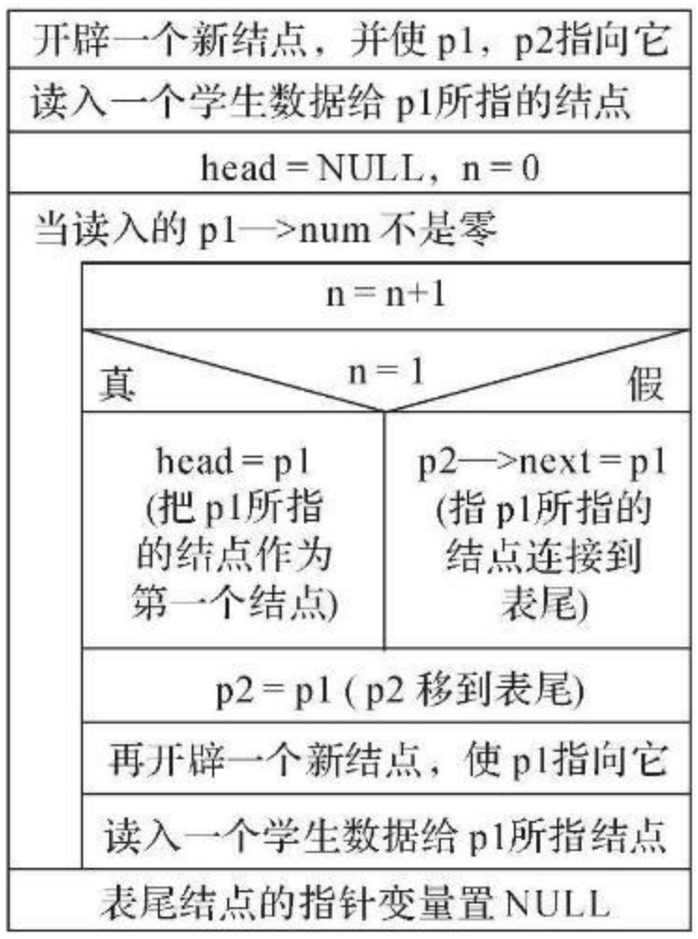


图 8 .12 建立链表的算法

链表头指针 head 的值为 NULL(即等于 0)，这是链表为“空”时的情况(即 head 不指向

更多高清PDF图书 www.gqpdf.com

高清PDF原创 www.gqpdf.com

任何结点,链表中无结点),以后增加一个结点就使 head 指向该结点。如果输入的 p1->num 不等于0,则输入的是第一个结点数据(n=1),令 head=p1,即把 p1 的值赋给 head,使 head 指向新开辟的结点(见图 8.13)。p1 所指向的新开辟的结点就成为链表中第一个结点。

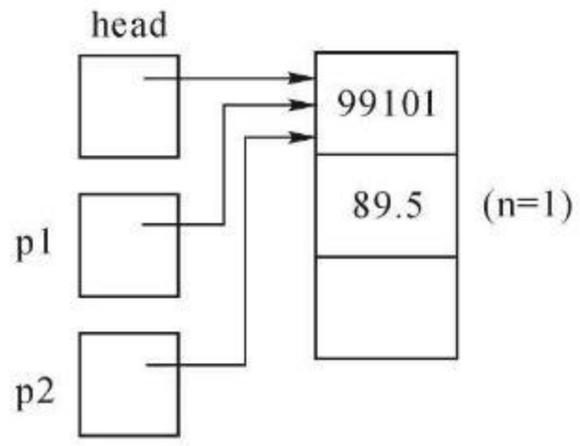


图 8.13　建立链表的第一个结点

然后再开辟另一个结点并使 p1 指向它,接着输入该结点的数据(见图 8.14(a))。如果输入的 p1->num ≠ 0,则应链入图 8.14 第 2 个结点(n=2),由于 n ≠ 1,则将 p1 的值赋给 p2->next,此时 p2 指向第一个结点,因此执行"p2->next=p1"就将新结点的地址赋给第一个结点的 next 成员,使第一个结点的 next 成员指向第二个结点(见图 8.14(b))。接着使 p2=p1,使 p2 指向刚才建立的结点(见图 8.14(c))。第三个结点的建立过程以此类推(插入第 3 个结点如图 8.15 所示)。

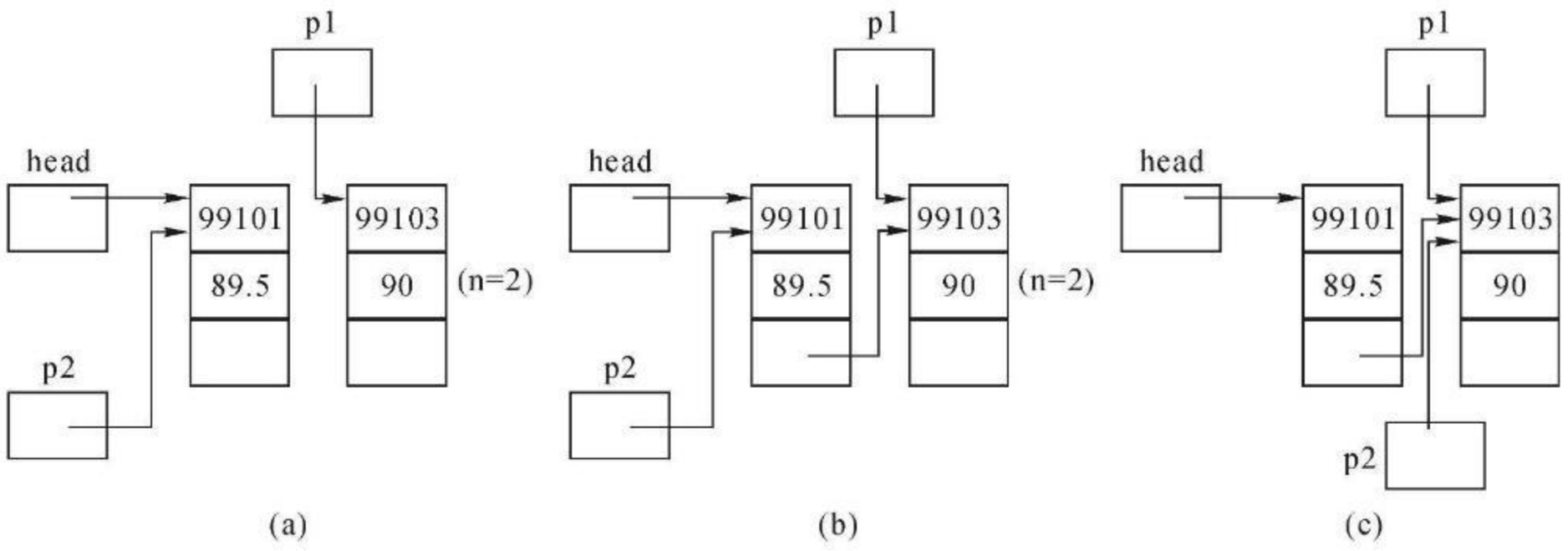


图 8.14　建立链表

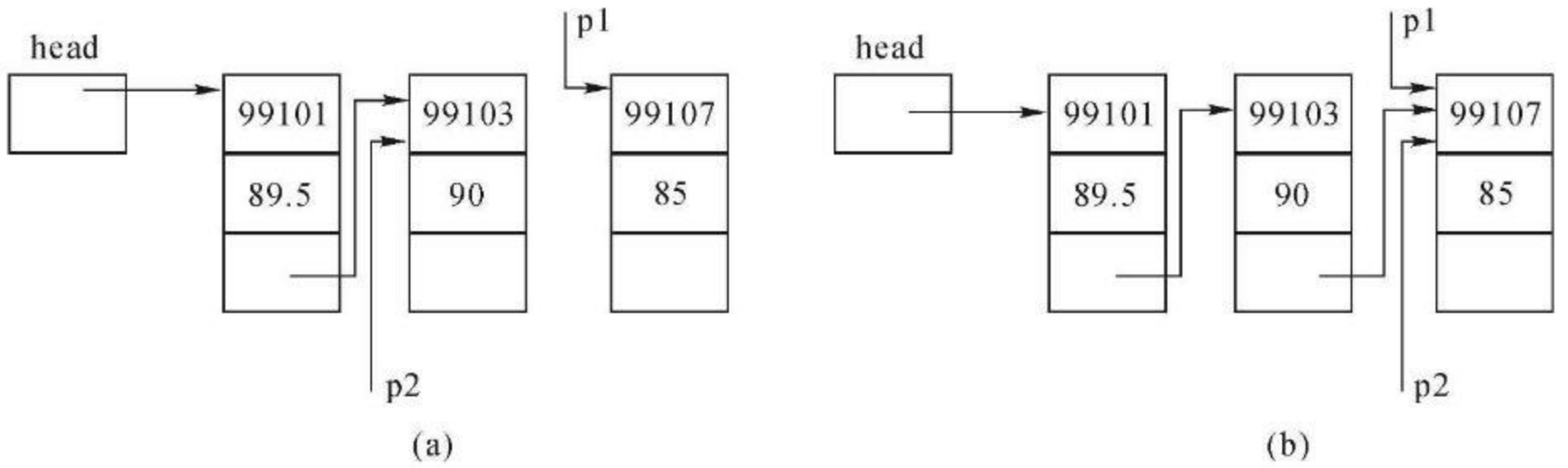


图 8.15　建立链表

再开辟一个新结点，并使 p1 指向它，输入该结点的数据(见图 8.16(a))。由于 p1->num 的值为 0，不再执行循环，此新结点不应被连接到链表中。此时将 NULL 赋给 p2->next(见图 8.16(b))。建立链表过程至此结束，p1 最后所指的结点未链入链表中，第 3 个结点的 next 成员的值为 NULL，它不指向任何结点。

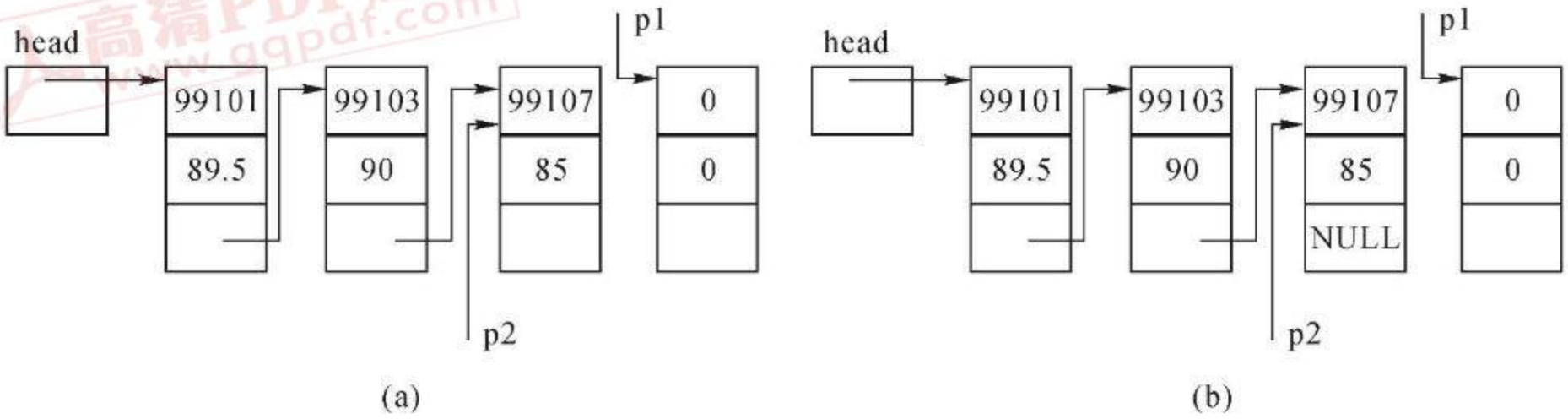


图 8.16 建立链表

建立链表的函数如下：

```
#define NULL 0
#define LEN sizeof(struct student)
#include <stdio.h>
#include <stdlib.h>
struct student
{
  long num;
  float score;
  struct student *next;
};

int n;  /* n为全局变量,本模块中各函数均可使用它 */

struct student *creat(void)  /*定义函数。此函数带回一个指向链表头的指针*/
{
  struct student *head;
  struct student *p1, *p2;

  n = 0;
  p1 = p2 = (struct student *)malloc(LEN);  /*开辟一个新单元*/
  scanf("%ld%f",&p1->num,&p1->score);
  head = NULL;

  while(p1->num != 0)
  {
    n = n + 1;
```

更多高清PDF图书 www.gqpdf.com

```
    if (n == 1)
      head = p1;
    else
      p2->next = p1;

    p2 = p1;
    p1 = (struct student * )malloc(LEN);
    scanf("%ld%f", &p1->num, &p1->score);
  }
  p2->next = NULL;

  return(head);
}
```

注意：

(1)第1行为#define命令，令NULL代表0，用它表示“空地址”。第2行令LEN代表struct student类型数据的长度，sizeof是“求字节数运算符”。

(2)第12行定义一个creat函数，函数返回值是指针类型，它指向一个struct student类型数据。creat函数返回一个链表起始地址。

(3)malloc(LEN)的作用是开辟一个长度为LEN的内存区，LEN已定义为sizeof(struct student)，即结构体struct student的长度。malloc返回的是不指向任何类型数据的指针(void * 类型)。而p1，p2是指向struct student类型数据的指针变量，因此必须用强制类型转换的方法使指针的基类型改变为struct student类型。在malloc(LEN)之前加了“(struct student *)”，作用是使malloc返回的指针转换为指向struct student类型数据的指针。“*”号不可省略，否则将变成转换成struct student类型，而不是指针类型了。

(4)最后一行return后面的参数是head(head已定义为指针变量，指向struct student类型数据)。因此函数返回的是head的值，即链表的头地址。

(5)n是结点个数。

8.5.4　链表的输出

将链表中各结点的数据依次输出，首先要知道链表第一个结点的地址，也就是要知道head的值。然后设一个指针变量p，先指向第一个结点，输出p所指的结点，然后使p后移一个结点，再输出，直到链表的尾结点。输出链表的算法见图8.17。

【例8.8】 编写一个输出链表的函数。

```
void print(struct student * head)
{
 struct student * p;

 printf("\nNow, These%d records are:\n", n);
 p=head;
```

```
if(head !=NULL)
  do
  {
      printf("%ld%5.1f\n",p->num,p->score);
      p=p->next;
  }
  while(p !=NULL);
}
```

说明:p 先指向第一结点(见图 8.18),在输出完第一个结点之后,p 移到图中 p' 虚线位置,指向第二个结点。程序中 p=p->next 的作用是将 p 原来所指向的结点中 next 的值赋给 p,而 p->next 的值就是第二个结点的起始地址。将它赋给 p,就是使 p 指向第二个结点。head 的值由实参传过来,将已有的链表的头指针传给被调用的函数,在 print 函数中从 head 所指的第一个结点出发顺序输出各个结点。

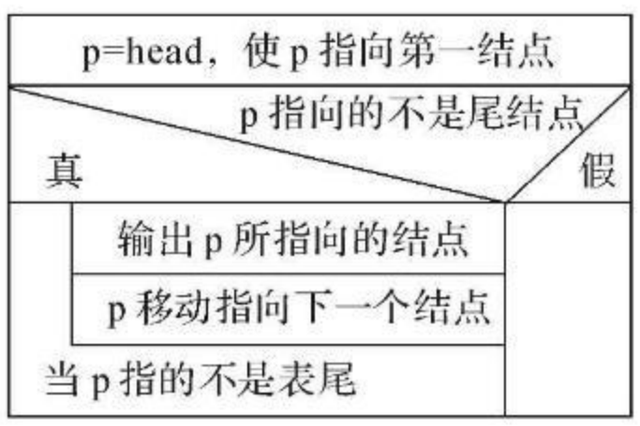


图 8.17 输出链表的算法

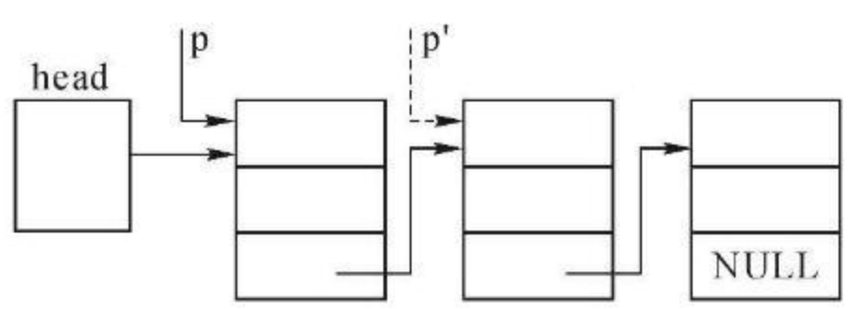


图 8.18 输出链表过程示意

8.5.5 对链表的删除操作

从一个链表中删去一个结点,并不是真正从内存中把它抹掉,而是把它从链表中分离开来,只要撤消原来的链接关系即可。

【例 8.9】 编写一个删除链表中指定的结点的函数。

以指定的学号作为删除结点的标志。例如,输入 99103 表示要求删除学号为 99103 的结点。

思路:从 p 指向的第一个结点开始,检查该结点中的 num 值是否等于输入的要求删除的那个学号。如果相等就将该结点删除,如不相等,就将 p 后移一个结点,再如此进行下去,直到遇到表尾为止。

设两个指针变量 p1 和 p2,先使 p1 指向第一个结点(图 8.19(a))。如果要删除的不是第一个结点,则使 p1 指向下一个结点(将 p1->next 赋给 p1),在此之前应将 p1 的值赋给 p2,使 p2 指向刚才检查过的那个结点(图 8.19(b))。如此一次一次地使 p1 后移,直到找到所要删除的结点或检查完整个链表都找不到要删除的结点为止。

如果找到某一个结点是要删除的结点,还要区分两种情况:

(1)要删的是第一个结点(p1 的值等于 head 的值,如图 8.19(a)),则应将 p1->next 赋给 head(图 8.19(c)),这时 head 指向原来的第二个结点。第一个结点虽然仍存在,但它已与链表脱离,因为链表中没有一个结点或头指针指向它。

更多高清PDF图书 www.gqpdf.com

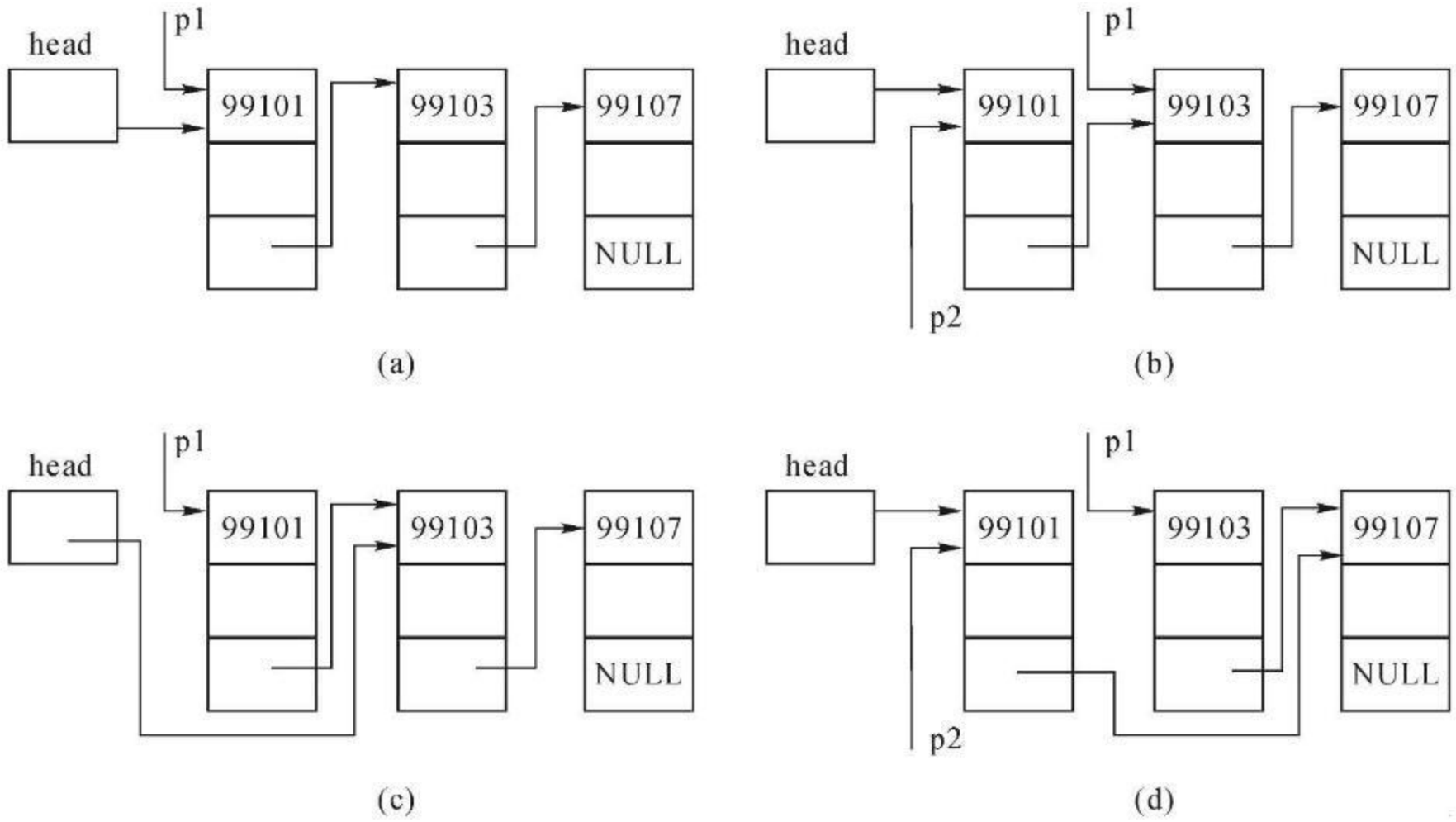


图 8.19　删除链表结点示意

(2)如果要删除的不是第一个结点,则将 p1->next 赋给 p2->next(图 8.19(d))。p2->next 原来指向 p1 指向的结点(图中第二个结点),现在 p2->next 改为指向 p1->next 所指向的结点(图中第三个结点)。p1 所指向的结点不再是链表的一部分。

另外,还需要考虑链表是否为空(无结点)和链表中找不到要删除的结点的情况。图 8.20 表示解此题的算法。

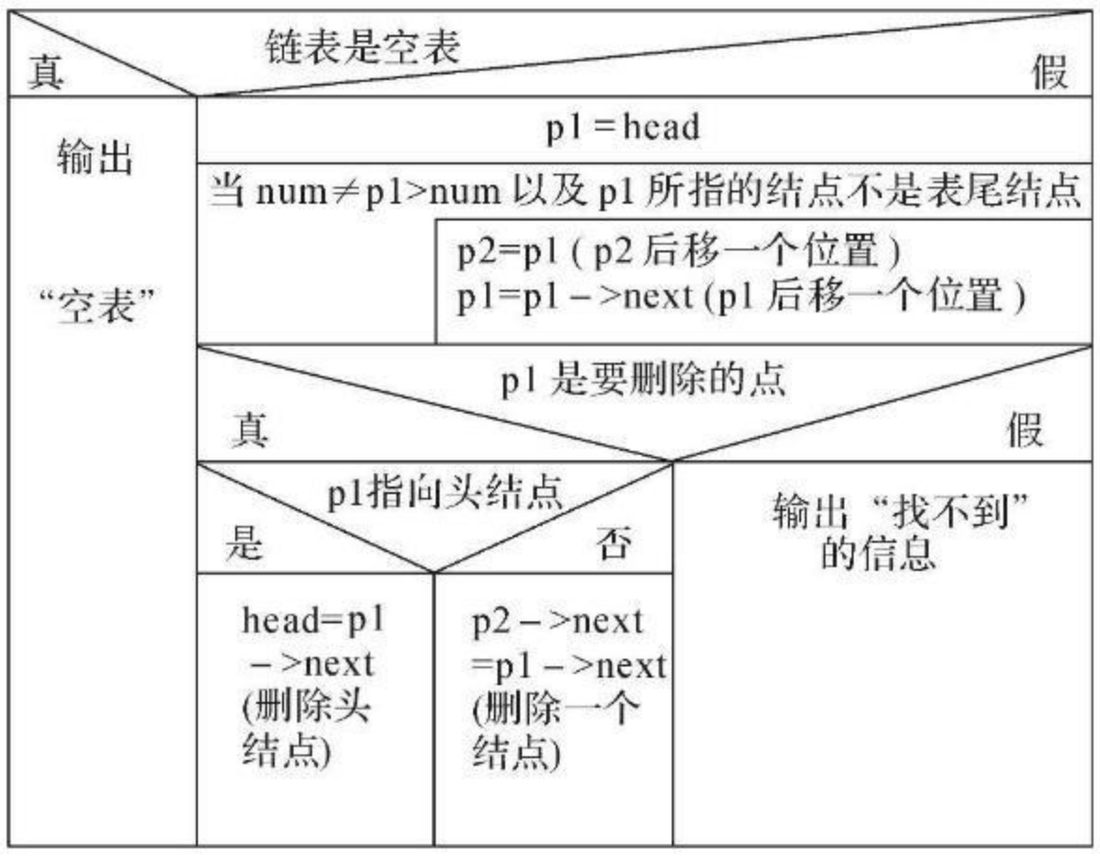


图 8.20　删除链表结点的算法

删除结点的函数如下:

```
struct student * del(struct student * head, long num)
{
struct student * p1, * p2;

if (head == NULL)
```

```
{
  printf("\nlist null!\n");

  return (head);
}

p1=head;
while(num !=p1->num && p1->next !=NULL)
    /*p1 指向的不是所要找的结点,并且后面还有结点*/
{
  p2=p1;
  p1=p1->next;
}    /*p1 后移一个结点*/

if(num == p1->num)     /*找到了*/
{
  if(p1 == head)
      head=p1->next;     /*若 p1 指向的是首结点,把第二个结点地址赋予 head*/
  else
      p2->next=p1->next;     /*否则将下一结点地址赋给前一结点地址*/

  printf("delete:%ld\n",num);
  n=n-1;
}
else
  printf("%ld not been found!\n",num);     /*找不到该结点*/

return (head);
}
```

函数返回值的类型是指向 struct student 类型数据的指针,它的值是链表的头指针。函数参数为 head 和要删除的学号 num。head 值可能在函数执行过程中被改变(当删除第一个结点时)。

8.5.6 链表结点的插入

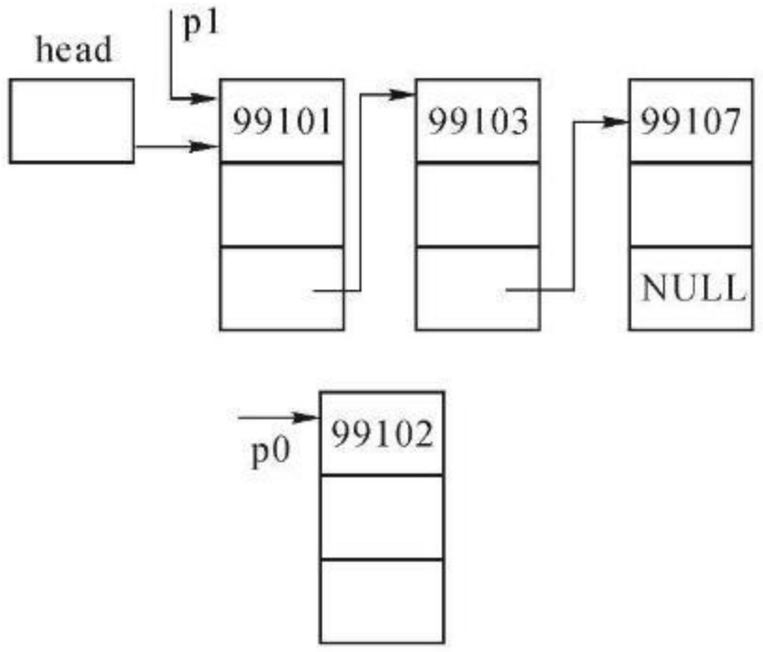
链表结点的插入是指将一个结点插入到一个已有的链表中。若已有一个学生链表,各结点是按其成员项 num(学号)的值由小到大顺序排列的。现在要插入一个新的结点,要求按学号的顺序插入。为了能正确插入,必须解决两个问题:①怎样找到插入的位置;②怎样实现插入。

如果有一群学生,按身高顺序(由低到高)手拉手排好队。现在来了一名新同学,要求按身高顺序插入队中。首先要确定插到什么位置。可以将新同学先与队中第 1 名小学生比身

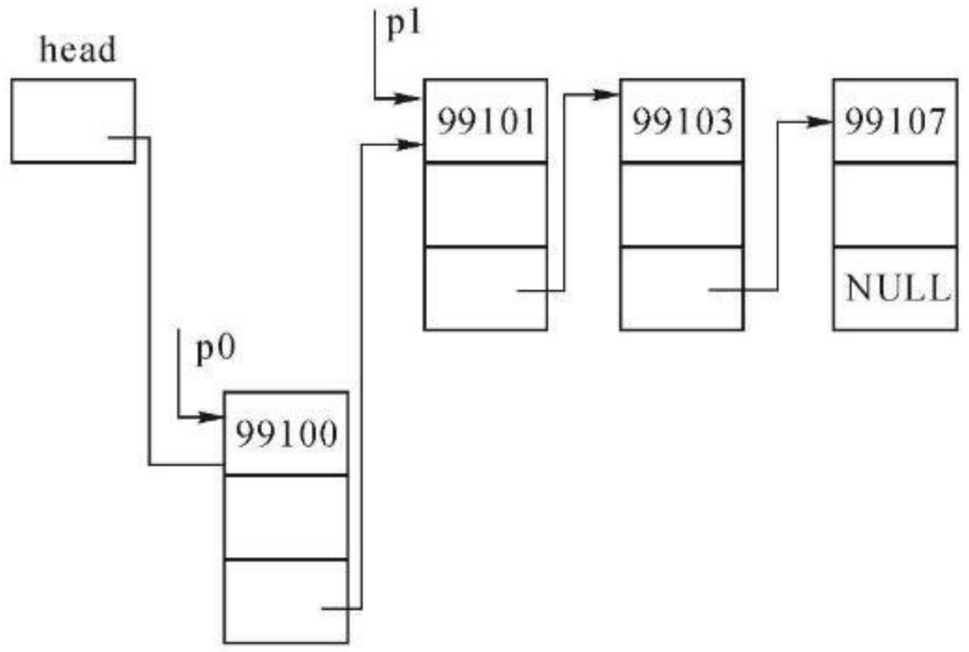
更多高清PDF图书 www.gqpdf.com

高,若新同学比第1名学生高,就使新同学后移一个位置,与第2名学生比,如果仍比第2名学生高,再往后移,与第3名学生比……直到出现比第i名学生高,比第i+1名学生低的情况为止。显然,新同学的位置应该在第i名学生之后,在第i+1名学生之前。在确定了位置之后,让第i名学生与第i+1名学生的手脱开,然后让第i名学生的手去拉新同学的手,让新同学另外一只手去拉第i+1名学生的手。这样就完成了插入,形成了新的队列。

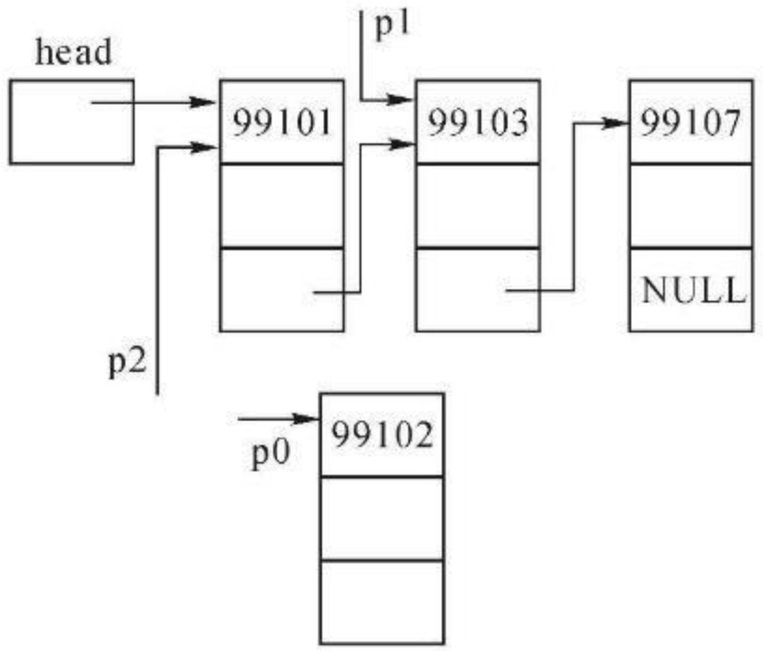
根据这个思路来实现链表的插入操作:先用指针变量p0指向待插入的结点,p1指向第一个结点(见图8.21(a))。将p0->num与p1->num相比较,如果p0->num>



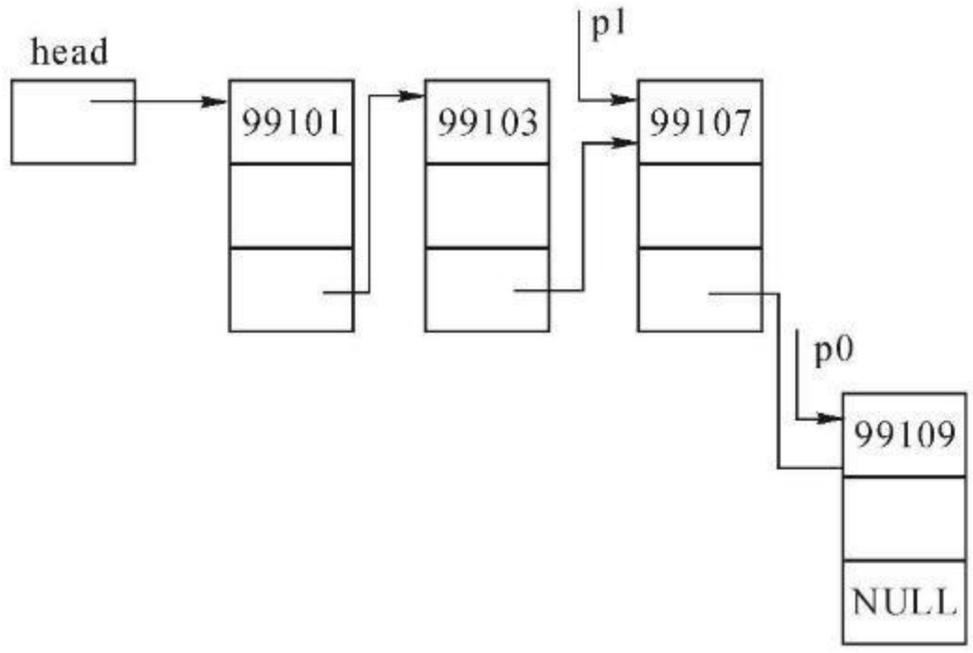
(a)



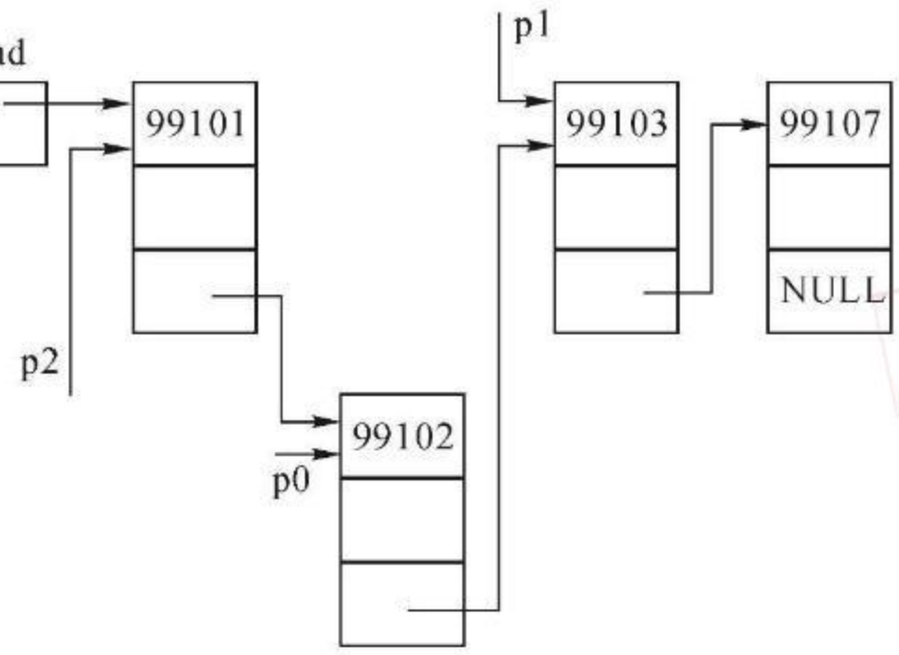
(b)



(c)



(d)



(e)

图8.21　链表结点的插入操作示意

高清PDF原创 www.gqpdf.com

p1->num,则待插入的结点不应插在 p1 所指的结点之前。此时将 p1 后移,并使 p2 指向刚才 p1 所指的结点(图 8.21(b))。再将 p1->num 与 p0->num 比……直到 p0->num ≤ p1->num 为止。这时将 p0 所指的结点插到 p1 所指结点之前。但是如果 p1 所指的已是表尾结点,则 p1 就不应后移了。如果 p0->num 比所有结点的 num 都大,则应将 p0 所指的结点插到链表末尾。

如果插入的位置既不在第一个结点之前,又不在表尾结点之后,则将 p0 的值赋给 p2->next,使 p2->next 指向待插入的结点,然后将 p1 的值赋给 p0->next,使得 p0->next指向 p1 指向的变量(图 8.21(c))。

如果插入位置为第一个结点之前(即 p1 等于 head 时),则将 p0 赋给 head,将 p1 赋给 p0->next(图 8.21(d))。

如果要插到表尾之后,则将 p0 赋给 p1->next,NULL 赋给 p0->next(图 8.21(e))。以上算法见图 8.22。

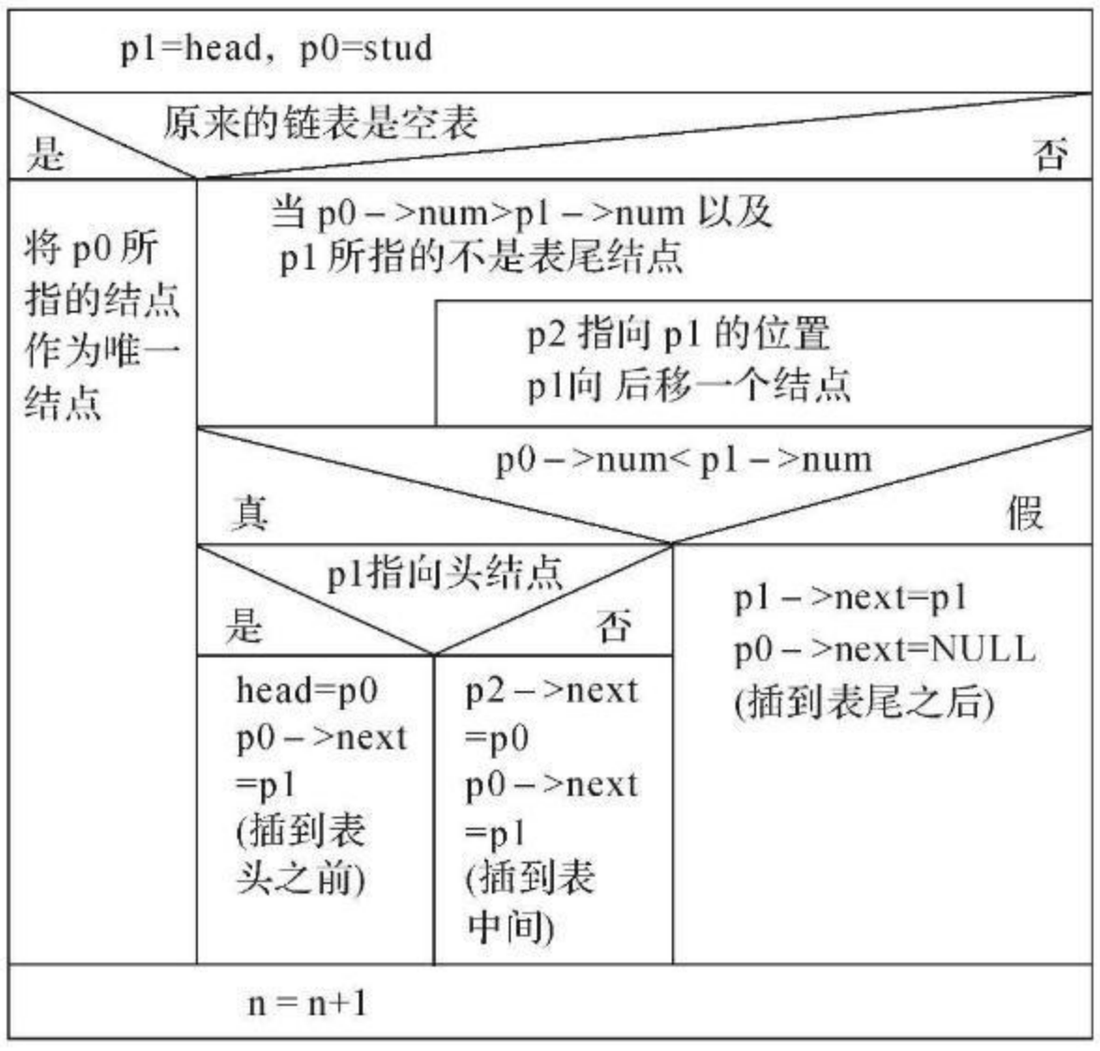


图 8.22 链表结点的插入操作算法

【例 8.10】 插入结点的函数。

```
struct student * insert(struct student * head, struct student * stud)
{
struct student * p0, * p1, * p2;

p1=head;                    /* 使 p1 指向第一个结点 */
p0=stud;                    /* p0 指向要插入的结点 */

if(head == NULL)            /* 原来的链表是空表 */
{
  head=p0;                  /* 使 p0 指向的结点作为头结点 */
```

高清PDF原创 www.gqpdf.com
更多高清PDF图书 www.gqpdf.com

```
    p0->next=NULL;
  }
  else
  {
    while((p0->num > p1->num)&& (p1->next !=NULL))
    {
        p2=p1;              /*使 p2 指向刚才 p1 指向的结点*/
        p1=p1->next;        /*p1 后移一个结点*/
    }

  if(p0->num < p1->num)
  {
      if(head == p1)
          head=p0;          /*插到原来第一个结点之前*/
      else
          p2->next=p0;/*插到 p2 指向的结点之后*/

      p0->next=p1;
  }
  else                      /*插到最后的结点之后*/
  {
          p1->next=p0;
          p0->next=NULL;
    }
  }

  n=n+1;                    /*结点数加 1*/

  return(head);
  }
```

函数参数是 head 和 stud。stud 是一个指针变量,从实参传来待插入结点的地址给 stud。语句 p0=stud;的作用是使 p0 指向待插入的结点。函数返回值类型是指针类型,函数返回值是链表起始地址 head。

8.5.7　链表的综合操作

将建立、输出、删除、插入的函数组织在一个 C 程序中,用 main 函数作主调函数。

```
voidmain()
{
 struct student *head, *stu;
 long del_num;
```

高清PDF原创 www.gqpdf.com

```
    printf("input records:\n");
    head=creat();
    print(head);

    printf("\ninput the deleted number: ");
    scanf("%ld",&del_num);
    while(del_num !=0)
    {
      head=del(head, del_num);
      print(head);
      printf("input the deleted number: ");
      scanf("%ld",&del_num);
    }

    printf("\ninput the inserted record: ");
    stu=(struct student *)malloc(LEN);
    scanf("%ld%f",&stu->num,&stu->score);
    while(stu->num !=0)
    {
      head=insert(head, stu);
      print(head);
      printf("input the inserted record: ");
      stu=(struct student *)malloc(LEN);
      scanf("%ld%f",&stu->num,&stu->score);
    }
    }
```

stu 定义为指针变量,在需要插入时先用 malloc 函数开辟一个内存区,将其起始地址经强制类型转换后赋给 stu,然后输入此结构体变量中各成员的值。对不同的插入对象,stu 的值是不同的,每次指向一个新的 struct student 变量。在调用 insert 函数时,实参为 head 和 stu,将已建立的链表起始地址传给 insert 函数的形参,将 stu(新开辟单元的地址)传给形参 stud,函数返回值是经过插入之后的链表的头指针。

8.6 共用体

8.6.1 共用体的定义

活动 2 共用体的概念。

有时需要使几种不同类型的变量存放到同一段内存单元中。例如,可把一个整型变量、

更多高清PDF图书 www.gqpdf.com

一个字符型变量、一个实型变量放在同一个地址开始的内存单元中(图8.23)。以上3个变量在内存中占的字节数不同,但都从同一个地址开始存放,也就是使用覆盖技术,几个变量互相覆盖。这种使几个不同的变量共同占用同一段内存的结构,称为“共用体”类型的结构。

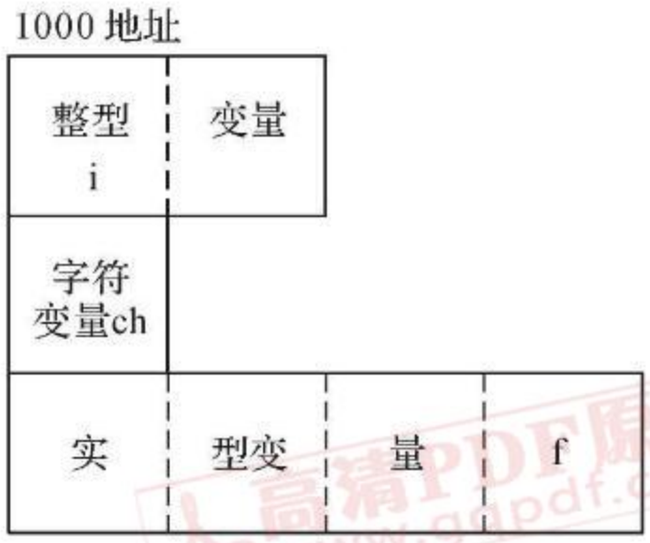


图 8.23　几种不同类型的变量存放到同一段内存单元中

8.6.2　共用体类型和变量的定义

定义共用体类型和变量的一般形式为

union 共用体名

{

/* 成员表列 */

类型 成员名

}变量表列;

例如:

```
union data
{
 int i;
char ch;
float f;
}a, b, c;
```

也可以将类型声明与变量定义分开:

```
union data
{
 int i;
char ch;
float f;
};
union data a, b, c;
```

也可以直接定义共用体变量:

```
union
{
 int i;
char ch;
float f;
}a, b, c;
```

“共用体”与“结构体”的定义形式相似。但它们的含义是不同的：

①结构体变量所占内存长度是各成员占的内存长度之和，每个成员分别占有其自己的内存单元；

②共用体变量所占的内存长度等于最长的成员的长度。

8.6.3 共用体变量的引用

与结构体类型变量一样，我们不能引用共用体变量，而只能引用共用体变量中的成员。例如，前面定义了 a，b，c 为共用体变量，下面的引用方式是正确的：

```
a.i    /*引用共用体变量中的类型变量 i*/
a.ch   /*引用共用体变量中的字符变量 ch*/
a.f    /*引用共用体变量中的实型变量 f*/
```

8.6.4 共用体类型数据的特点

(1)同一个内存段可以用来存放几种不同类型的成员，但在同一时刻只能存放其中一种，而不是同时存放几种。即同一时刻只有一个成员存在，其他的成员不存在。

(2)共用体变量中起作用的成员是最后一次存放的成员，在存入一个新的成员后原有的成员就失去作用。如以下赋值语句：

```
a.i=1;
a.ch='a';
a.f=1.5;
```

在完成以上 3 个赋值运算以后，只有 a.f 是有效的，a.i 和 a.ch 已经无意义了。此时只能用 printf("%f", a.f)。在引用共用体变量时应十分注意当前存放在共用体变量中的究竟是哪个成员。

(3)共用体变量的地址和它的各成员的地址都是同一地址。

(4)不能对共用体变量赋值，不能企图引用变量名来得到一个值，不能在定义共用体变量时对它初始化。

(5)不能把共用体变量作为函数参数，也不能使函数带回共用体变量，但可以使用指向共用体变量的指针。

(6)共用体类型可以出现在结构体类型定义中，也可以定义共用体数组。反之，结构体也可以出现在共用体类型定义中，数组也可以作为共用体的成员。

8.6.5 共用体应用举例

【例 8.11】 设有若干个人员的数据，其中有学生和教师。学生的数据中包括：姓名、号码、性别、职业、班级。教师的数据包括：姓名、号码、性别、职业、职务。学生和教师所包含的数据是不同的。现要求把它们放在同一表格中(图 8.24)。如果“job”项为“s”(学生)，则第

更多高清PDF图书 www.gqpdf.com

5项为class(班)。即Li 501班的。如果“job”项是“t”(教师),则第5项为position(职务)。Wang是prof(教授)。

name	num	sex	job	class (班) / position (职务)
Li	1011	f	s	501
wang	2085	m	t	prof

图8.24　例8.11的示意

对第5项可以用共用体来处理(将class和position放在同一段内存中)。要求输入人员的数据,然后再输出。算法见图8.25。

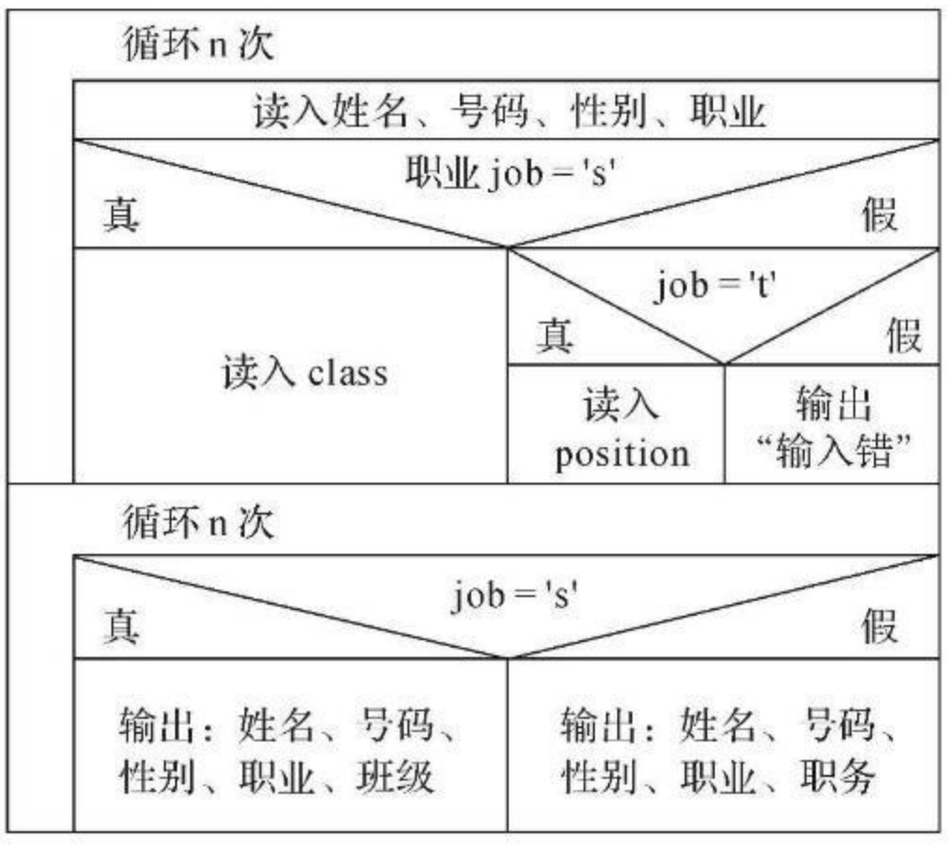


图8.25　例8.11的算法

```
#include<stdio.h>

struct
{
int num;
char name[10];
char sex;
char job;
union
{
  int Class;
  char position[10];
} category;
} person[2];

void main()
```

```
{
int i;

for(i=0; i<2; i++)
{
  scanf("%d%s%d%c",&person[i].num,person[i].name,&person[i].sex,&person[i].job);
  if(person[i].job == 's')
      scanf("%d",&person[i].category.Class);
  else
      if(person[i].job == 't')
          scanf("%s",person[i].category.position);
      else
          printf("input error!");
}
printf("\n");
printf("No.Namesex job class/position\n");

for(i=0; i<2; i++)
{
  if(person[i].job == 's')
      printf("%-6d%-10s%-3c%-3c%-6d\n", person[i].num, person[i].name,
      person[i].sex, person[i].job, person[i].category.Class);
  else
      printf("%-6d%-10s%-3c%-3c%-6s\n", person[i].num, person[i].name,
      person[i].sex, person[i].job, person[i].category.position);
}
}
```

*8.7 枚举类型

“枚举”是指将变量所有可能的值一一列举出来,变量的值只限于列举出来的值的范围内。声明枚举类型用 enum 开头。例如:

```
enum weekday{sun,mon, tue,wed, thu, fri, sat};
```

声明了一个枚举类型 enum weekday,可以用此类型来定义变量。如:

```
enum  weekday workday,week_end;
```

workday 和 week_end 被定义为枚举变量,它们的值只能是 sun 到 sat 之一。例如:

```
workday=mon;
week_end= sun;
```

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

也可以直接定义枚举变量，如：

```
enum{sun,mon, tue,wed, thu, fri, sat}workday,week_end;
```

其中的 sun，mon，…，sat 等称为枚举元素或枚举常量。它们是用户定义的标识符。这些标识符并不自动地代表什么含义。用什么标识符代表什么含义，完全由程序员决定，并在程序中作相应处理。

说明：

(1)C 语言编译时，对枚举元素按常量处理，故称枚举常量。它们不是变量，不能对它们赋值。

(2)枚举元素作为常量，它们是有值的，C 语言编译按定义时的顺序使它们的值为 0，1，2，…

在上面定义中，sun 的值为 0，mon 的值为 1……sat 为 6。如果有赋值语句：

```
workday=mon;
```

workday 变量的值为 1。这个整数是可以输出的。如：

```
printf("%d",workday);
```

将输出整数 1。

也可以改变枚举元素的值，在定义时由程序员指定，如：

```
enum weekday{sun=7,mon=1, tue,wed, thu, fri, sat}workday,week_end;
```

定义 sun 为 7，mon=1，以后顺序加 1，sat 为 6。

(3)枚举值可以用来做判断比较。如：

```
if(workday == mon)
if(workday > sun)
```

枚举值的比较规则是按其在定义时的顺序号比较。

(4)一个整数不能直接赋给一个枚举变量。如：

```
workday=2;
```

是不对的。它们属于不同的类型。应先进行强制类型转换才能赋值。如：

```
workday=(enum weekday)2;
```

相当于将顺序号为 2 的枚举元素赋给 workday，相当于：

```
workday= tue;
```

也可以是表达式。如：

```
workday=(enum weekday)(5-3);
```

【例 8.12】 口袋中有红、黄、蓝、白、黑 5 种颜色的球若干个。每次从口袋中先后取出 3 个球，问得到 3 种不同色的球的可能取法，打印出每种排列的情况。

球只是 5 种色之一，而且要判断各球是否同色，应该用枚举类型变量处理。设取出的球为 i，j，k。根据题意，i，j，k 分别是 5 种色球之一，并要求 i ≠ j ≠ k。可以用穷举法，即一

种可能一种可能地试,看哪一组符合条件。算法见图 8.26。

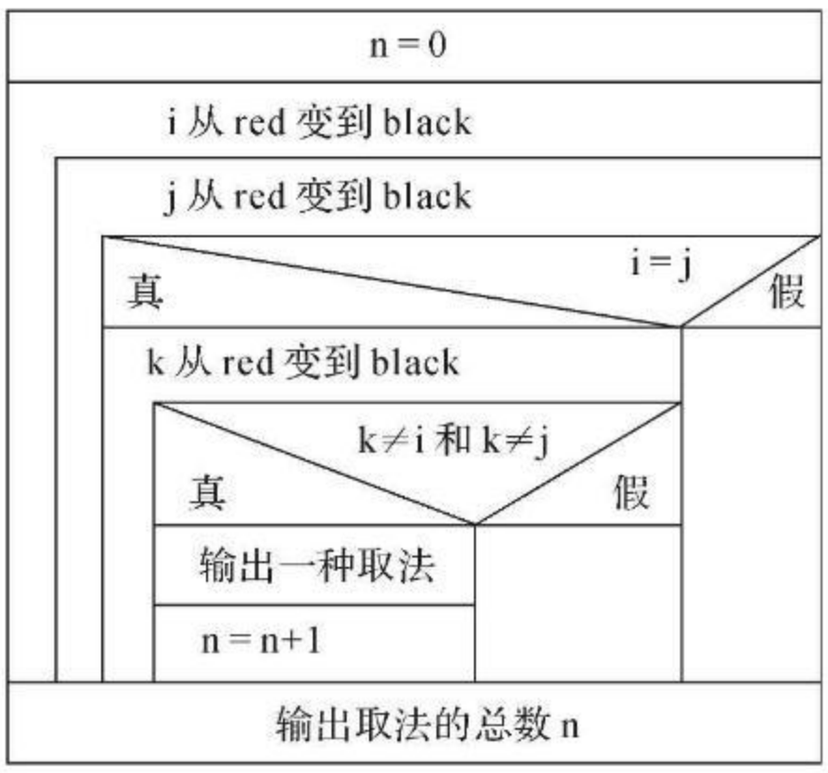


图 8.26 例 8.12 算法

用 n 累计得到 3 种不同色球的次数。第一层循环使第 1 个球 i 从 red 变到 black。第二层循环使第 2 个球 j 也从 red 变到 black。如果 i 和 j 同色则不可取,只有 i, j 不同色(i ≠ j)时才需要继续找第 3 个球,此时第 3 个球 k 也有 5 种可能(red 到 black),但要求第 3 个球不能与第 1 个球或第 2 个球同色,即 k ≠ i, k ≠ j。满足此条件就得到 3 种不同色的球。输出这种 3 色组合方案。然后使 n 加 1。外循环全部执行完后,全部方案就已输出完了。最后输出总数 n。

为了输出 3 个球的颜色,应经过 3 次循环,第 1 次输出 i 的颜色,第 2 次输出 j 的颜色,第 3 次输出 k 的颜色。在 3 次循环中先后将 i, j, k 赋予 pri。然后根据 pri 的值输出颜色信息。在第 1 次循环时,pri 的值为 i,如果 i 的值为 red,则输出字符串“red”,其他的类推(见图 8.27)。

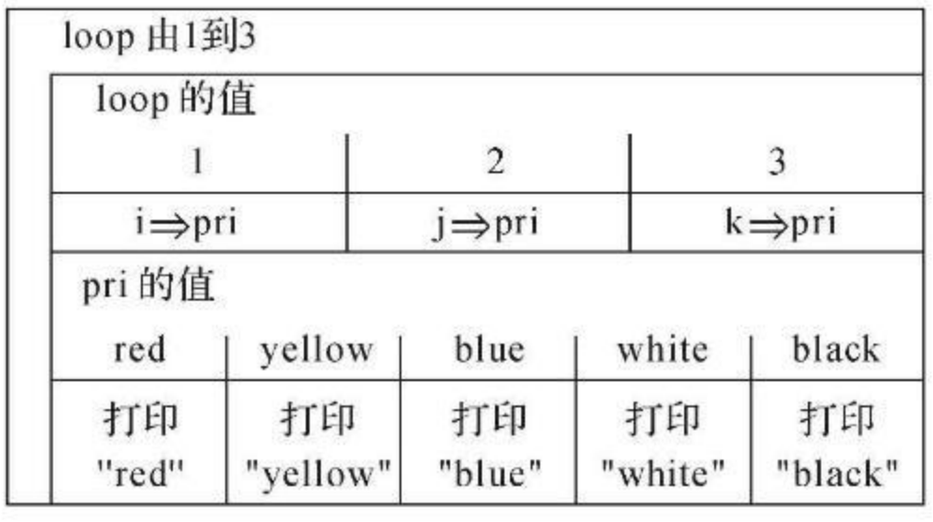


图 8.27 输出一种取法

```
#include<stdio.h>

voidmain()
{
enum color{red,yellow,blue,white,black};

enum color i,j,k,pri;
```

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

```
    int n, loop;

    n=0;

    for(i=red; i<=black; i++)
      for(j=red; j<=black; j++)
          if(i !=j)
          {
              for(k=red; k<=black; k++)
                  if((k !=i)&& (k !=j))
                  {
                      n=n+1;
                      printf("%-4d", n);

                      for(loop=1; loop<=3; loop++)
                      {
                          switch(loop)
                          {
                          case 1: pri=i; break;
                          case 2: pri=j; break;
                          case 3: pri=k; break;
                          default: break;
                          }

                          switch(pri)
                          {
                          case red: printf("%-10s", "red"); break;
                          case yellow: printf("%-10s", "yellow"); break;
                          case blue: printf("%-10s", "blue"); break;
                          case white: printf("%-10s", "white"); break;
                          case black: printf("%-10s", "black"); break;
                          default: break;
                          }
                      }
                      printf("\n");
                  }
          }
          printf("\ntotal:%5d\n", n);
    }
```

8.8　本章小结

结构体使得零散的数据项能集中在一个整体中管理，使得对数据对象的管理更直观。

这也是面向对象程序设计中类的前身,是比较重要的数据类型。链表的运用很广泛,尤其在“数据结构”课程中更是常用。链表对初学者来说比较抽象,对链表的操作掌握起来比较困难,要通过多看程序、辅助画图来理解,多编程来巩固。

课后练习

1.有 10 个学生,每个学生的数据包括学号,姓名,3 门课的成绩,从键盘输入 10 个学生数据,要求打印出 3 门课总平均成绩,以及最高分的学生的数据(包括学号,姓名,3 门课的成绩,平均分数)。

2.建立一个链表,每个结点包括:学号、姓名、性别、年龄。输入一个年龄,如果链表中的结点所包含的年龄等于此年龄,则将此结点删去。

更多高清PDF图书 www.gqpdf.com

第 9 章

位运算

高清PDF原创 www.gaopdf.com

【本章主要内容】

位运算是指进行二进制位的运算。在计算机用于检测和控制的领域以及系统软件中，常要处理二进位的问题。例如，将一个存储单元中的各二进位左移或右移一位，两个数按位相加等。C 语言提供了位运算的功能，与其他高级语言相比，它显然具有很大的优越性。

【本章要求】

(1)了解位运算符的含义和使用。
(2)掌握简单的位运算。

9.1 位运算符和位运算

C 语言提供的位运算符如表 9.1 所示。

表 9.1 C 语言提供的位运算符

运算符	含 义	运算符	含 义
&	按位与	~	按位取反
\|	按位或	<<	左移
^	按位异或	>>	右移

9.1.1 按位与“&”运算符

参加运算的两个数，按二进位进行“与”运算。如果两个数相应的二进位都为 1，则该位

的结果值为 1,否则为 0。即

0 & 0=0;　　0 & 1=0;　　1 & 0=0;　　1 & 1=1

例如:3 & 5

```
        3=00000011
  &     5=00000101
  ------------------
        1=00000001
```

因此,3 & 5 的值得 1。如果参加 & 运算的是负数(如 -3 & -5),则以补码形式表示为二进制数,然后按位进行“与”运算。

活动 1　思考按位与 & 运算符的用途。

(1)清零。如果想将一个单元清零,即使其全部二进位为 0,只要用一个全 0 的数和其进行与运算即可。

如:原数为 00101011:

```
        00101011
  &     00000000
  ------------------
        00000000
```

(2)取一个数中某些指定位。如有一个整数 a(2 个字节),想要其中的低字节。只需将 a 与 (737),即按位与即可(图 9.1)。

如果想取两个字节中的高字节,只需 c=a & 0177400　(0177400 表示八进制数的 177400)。见图 9.2。

a	00 10 11 00	10 10 11 00
b	00 00 00 00	11 11 11 11
c	00 00 00 00	10 10 11 00

图 9.1　取一个数中的低字节

a	00 10 11 00	10 10 11 00
b	11 11 11 11	00 00 00 00
c	00 10 11 00	00 00 00 00

图 9.2　取一个数中的高字节

(3)要想将哪一位保留下来,就与一个数进行 & 运算,此数在该位取 1。如:有一数 01010100,想把其中第 3,4,5,7,8 位保留下来:

```
        01010100        十进制数 84
  &     00111011        十进制数 59
  ------------------
        00010000        十进制数 16
```

更多高清PDF图书　www.gqpdf.com

9.1.2　按位或“|”运算符

参加运算的两个数,按二进位进行“或”运算。两个数相应的二进位中只要有一个为1,该位的结果值为1。即

0|0=0;　　0|1=1;　　1|0=1;　　1|1=1

例如:060|017将八进制数60与八进制数17进行按位或运算。

```
  00110000
| 00001111
----------
  00111111
```

按位或运算常用来对一个数据的某些位定值为1。如:a是一个整数(16位),有表达式a|0377则低8位全置为1。高8位保留原样。

9.1.3　异或“^”运算符

异或运算符^也称XOR运算符。规则是若参加运算的两个二进位:同号则结果为0(假);异号则结果为1(真)。即

0^0=0;　　0^1=1;　　1^0=1;　　1^1=0

如:

```
  00111001   十进制数57,八进制数071
^ 00101010   十进制数42,八进制数052
  00010011   十进制数19,八进制数023
```

即071^052,结果为023(八进制数)。

“异或”是判断两个相应的位值是否为“异”,为“异”(值不同)就取真(1),否则取假(0)。

活动2　思考异或^运算符的用途。

(1)使特定位翻转

要使哪几位翻转就将与其进行^运算的数的几位置为1即可。这是因为原数中值为1的位与1进行^运算得0,原数中的位值0与1进行^运算的结果得1。

如:有01111010,想使其低4位翻转,即1变为0,0变为1。可以将它与00001111进行^运算,即

```
  01111010
^ 00001111
----------
  01110101
```

因为原数中的1与1进行^运算得0,0与1进行^运算得1。

(2)保留原值

要使哪几位保留原值就将与其进行 ^ 运算的数的几位置为 0 即可,如:

```
  00001010
^ 00000000
----------
  00001010
```

因为原数中的 1 与 0 进行 ^ 运算得 1,0 与 0 进行 ^ 运算得 0。

(3)交换两个值,不用临时变量

假如 a=3,b=4。想将 a 和 b 的值互换,可以用以下赋值语句实现:

```
a=a^b;
b=b^a;
a=a^b;
```

用以下竖式说明:

```
  a=011
^ b=100
-------
  a=111    a^b 的结果,a 已变成 7
^ b=100
-------
  b=011    b^a 的结果,b 已变成 3
^ a=111
-------
  a=100    a^b 的结果,a 变成 4
```

即等效于以下两步:

① b=b^(a^b)=b^a^b=a^b^b=a^0=a

相当于上面的前两个赋值语句:a=a^b; 和 b=b^a; b^b 的结果为 0,因为同一个数与本身相 ^,结果必为 0。现在 b 已得到 a 的值 3。在上式中除了第一个 b 以外,其余的 a,b 都是指原来的 a,b。

② a=a^b=(a^b)^(b^a^b)=a^b^b^a^b=a^a^b^b^b=b

a 得到 b 原来的值。

9.1.4 按位取反“~”运算符

~ 是一个单目(元)运算符,用来对一个二进制数按位取反,即将 0 变 1,1 变 0。例如:~ 025 是对八进制数 25 按位取反。

```
~  000000000010101
---------------
  111111111101010
```

即八进制数 177752。因此,~ 025 的值为八进制数 177752。不要以为 ~ 025 的值是 -025。

更多高清PDF图书 www.gqpdf.com

高清PDF原创 www.gqpdf.com

～运算符的应用：

若一个整数a为16位，想使最低一位为0，可以用：

a=a & 0177776

177776即二进制数1111111111111110，如果a的值为八进制数75，a & 0177776的运算可以表示如下：

```
  0000000000111101
& 1111111111111110
------------------
  0000000000111100
```

a的最后一个二进位变成0。但如果将C源程序移植到以32位存放一个整数的计算机系统上，由于一个整数用4个字节(32位)表示，想将最后一位变成0就不能用a & 0177776了。

为了适应以32位存放一个整数的计算机系统，应改用：

a & 037777777776

这样改动使移植性差了。改用：

a=a & ～1

它对以16位和以32位存放一个整数的情况都适用，不必作修改。因为在以2个字节存储一个整数时，1的二进制形式为0000000000000001，～1是1111111111111110。在以4个字节存储一个整数时，～1是11111111111111111111111111111110。

～运算符的优先级别比算术运算符、关系运算符、逻辑运算符和其他位运算符都高。

9.1.5　左移“<<”运算符

用来将一个数的各二进位全部左移若干位。例如：

a=a << 2

将a的二进制数左移2位，右补0。若a=15，即二进制数00001111，左移2位得00111100，即十进制数60。高位左移后溢出，舍弃不起作用。

左移1位相当于该数乘以2，左移2位相当于该数乘以22=4。但此结论只适用于该数左移时被溢出舍弃的高位中不包含1的情况。左移比乘法运算快得多，有些C编译程序自动将乘2的运算用左移一位来实现，将乘2^n的幂运算处理为左移n位。

9.1.6　右移“>>”运算符

用来将一个数的各二进位全部右移若干位。例如：

a=a >> 2

将a的二进制数右移2位。移到右端的低位被舍弃，对无符号数，高位补0。如a=017时：a为00001111，a >> 2为00000011。右移一位相当于除以2，右移n位相当于除以2^n。

在右移时，需要注意符号位问题。对无符号数，右移时左边高位移入0。对于有符号的值，如果原来符号位为0(该数为正)，则左边也是移入0。如果符号位原来为1(即负数)，则

左边移入 0 还是 1,要取决于所用的计算机系统。有的系统移入 0(“逻辑右移”),有的移入 1(“算术右移”)。

例如,a 的值为八进制数 113755:

a: 1001011111101101

a >> 1: 0100101111110110(逻辑右移时)

a >> 1: 1100101111110110(算术右移时)

9.1.7 位运算赋值运算符

位运算符与赋值运算符可以组成复合赋值运算符。

如:&=, |=, >>=, <<=, ^=

例如:a &=b 相当于 a=a & b

a <<=2 相当于 a=a << 2。

9.1.8 不同长度的数据进行位运算

如果两个数据长度不同进行位运算时,系统会将二者按右端对齐。

如果为正数,则左端补满 0。

如果为负数,则左端补满 1。

如果为无符号型,则左端补满 0。

9.2 位运算举例

【例 9.1】 取一个整数 a 从右端开始的 4~7 位。

算法:

①先使 a 右移 4 位(图 9.3)。图 9.3(a) 是未右移时的情况,图 9.3(b)是右移 4 位后的情况。目的是使要取出的那几位移到最右端。



(a) (b)

图 9.3 取整数的 4~7 位

右移到右端可以用下面方法实现:

a >> 4

②设置一个低 4 位全为 1,其余全为 0 的数。可用下面方法实现:

~(~ 0 << 4)

~ 0 的全部二进制为全 1,左移 4 位,这样右端低 4 位为 0。如下所示:

更多高清PDF图书 www.gqpdf.com

高清PDF原创 www.gqpdf.com

0：　0000…000000
～0：　1111…111111
～0<<4：　1111…110000
～(～0<<4)：　0000…001111

③将上面二者进行 & 运算。即

(a>>4) & ～(～0<<4)

```
#include "stdio.h"
voidmain()
{
  unsigned a, b, c, d;
  scanf("%o", &a);
  b=a>>4;
  c=～(～0<<4);
  d=b& c;
  printf("%o,%d\n%o,%d\n", a, a, d, d);
}
```

可以任意指定从右面第 m 位开始取其右面 n 位。只需将程序中的“b=a>>4”改成“b=a>>(m－n+1)”以及将“c=～(～0<<4)”改成“c=～(～0<<n)”即可。

【例 9.2】 循环移位。要求将 a 进行右循环移位(图 9.4)。即将 a 中原来左面 (16－n) 位右移 n 位,原来右端 n 位移到最左面 n 位。今假设用两个字节存放一个整数。为实现以上目的可以用以下步骤：

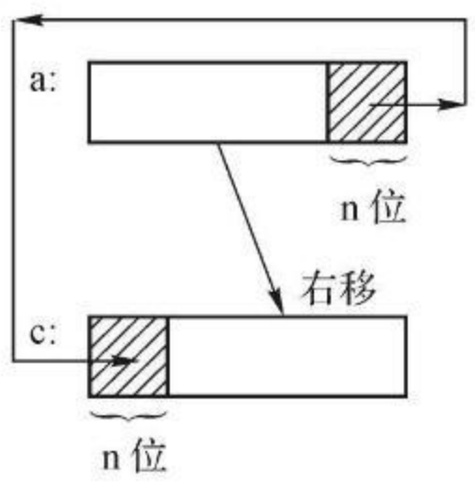


图 9.4　循环右移

①将 a 的右端 n 位先放到 b 中的高 n 位中：

b=a<<(16－n);

②将 a 右移 n 位,其左面高位 n 位补 0：

c=a>>n;

③将 c 与 b 进行按位或运算：

c=c|b;

```
#include "stdio.h"
```

```
voidmain()
{
  unsigned a, b, c;
  int n;
  scanf("%o%d", &a, &n);
  b=a<<(16-n);
  c=a>>n;
  c=c|b;
  printf("%o\n%o\n", a, c);
}
```

同样可以实现左循环位移。

高清PDF原创 www.gqpdf.com

9.3 位 段

对内存中信息的存取一般以字节为单位,有时存储一个信息不必用一个或多个字节,例如,“真”或“假”用 0 或 1 表示,只需 1 位即可。

向一个字节中的一个或几个二进位赋值和改变它的值的方法:

(1)人为地在一个字节 data 中设几项。例如:a, b, c, d 分别占 2 位、6 位、4 位、4 位(图 9.5)。如果想将 c 的值变为 12(设 c 原来为 0):

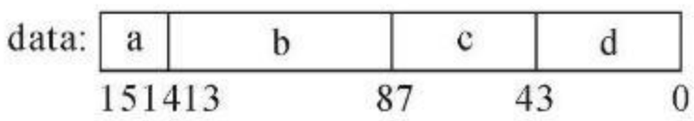


图 9.5 位段的示意

①将数 12 左移 4 位,使 1100 成为右面起第 4~7 位。

②将 data 与“12<<4”进行“按位或”运算,即可使 c 的值变成 12。

如果 c 的原值不为 0,应先使之为 0:

data=data & 0177417

0177417 的二进制表示为:

11	111111	0000	1111
a	b	c	d

也就是使第 4~7 位全为 0,其他位全为 1。它与 data 进行 & 运算,使第 4~7 位为 0,其余各位保留 data 的原状。这个 177417 称为“屏蔽字”,即把 c 以外的信息屏蔽起来,不受影响,只使 c 改变为 0。但要找出和记住屏蔽字比较麻烦。可以用:

```
data=data&~(15<<4);
```

15 是 c 的最大值,c 共占 4 位,最大值为 1111 即 15。15<<4 是将 1111 移到 4~7 位。再取反,就使 4~7 位变成 0,其余位全是 1。即

更多高清PDF图书 www.gqpdf.com

原创PDF原创 www.yuapdf.com

15：　0000000000001111
15 << 4：　0000000011110000
～(15 << 4)：　1111111100001111

这样可以实现对 c 清 0，而不必计算屏蔽码。

将上面几步结合起来，可以得到：

```
data=data& ～(15 << 4) |(n& 15)<< 4;
```

赋予 4～7 位为 0。

n 为应赋给 c 的值。n & 15 的作用是只取 n 的右端 4 位的值，其余各位置 0，即把 n 放到最后 4 位上，(n & 15) << 4，就是将 n 置在 4～7 位上。见下面：

data & ～(15<<4)：	11011011	0000	1010
(n & 15)<<4：	00000000	1100	0000
(按位或运算)	11011011	1100	1010

data 的其他位保留原状未改变，而第 4～7 位改变为 12(即 1100)了。

但是用以上办法给一个字节中某几位赋值很不方便。

(2)位段

C 语言允许在一个结构体中以位为单位来指定其成员所占内存长度。以位为单位的成员称为“位段”或称“位域”(bit field)。利用位段能够用较少的位数存储数据。

例如：

```
struct packed_data
{
  unsigned a: 2;
  unsigned b: 6;
  unsigned c: 4;
  unsigned d: 4;
  int i;
} data;
```

其中 a，b c，d 分别占 2 位、6 位、4 位、4 位(图 9.6)。i 为整型。共占 4 个字节。

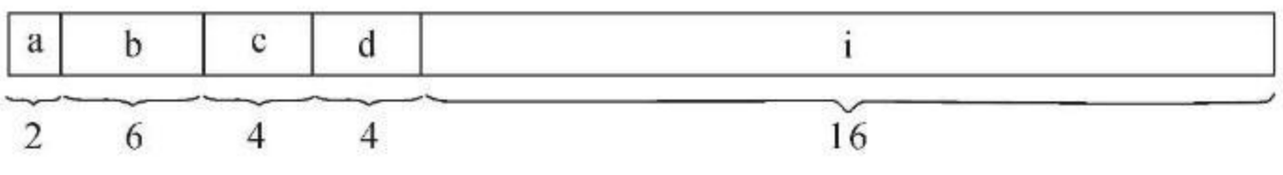


图 9.6　定义一个位段

各个位段可以不恰好占满一个字节。如：

```
struct packed_data
{
  unsigned a: 2;
  unsigned b: 3;
  unsigned c: 4;
  int i;
```

```
};
struct packed_data data;
```

其中 a，b，c 共占 9 位，占 1 个字节多，不到 2 个字节。它的后面为 int 型，占 2 个字节。在 a，b c 之后 7 位空间闲置不用，i 从另一字节开头起存放(图 9.7)。

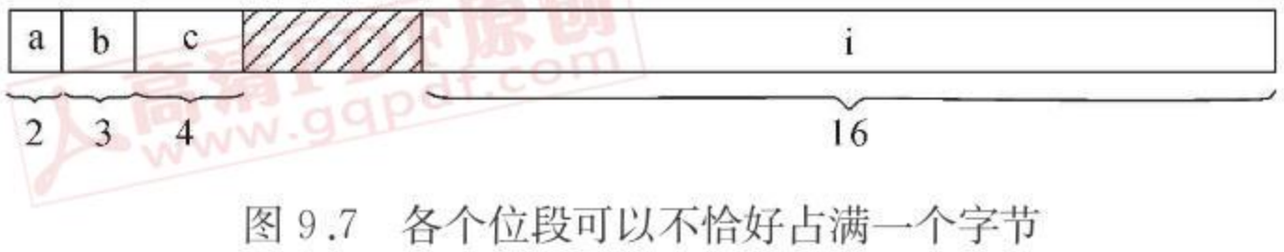


图 9.7 各个位段可以不恰好占满一个字节

在存储单元中位段的空间分配方向，因机器而异。在微机使用的 C 系统中，一般是由右到左进行分配的(图 9.8)。

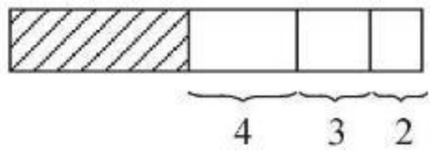


图 9.8 存储单元中位段的空间分配方向

对位段中的数据引用的方法：

```
data.a=2;
data.b=7;
data.c=9;
```

注意位段允许的最大值范围。如果写：

```
data.a=8;
```

就错了。因为 data.a 只占 2 位，最大值为 3。在此情况下，自动取赋予它的数的低位。例如，8 的二进制数形式为 1000，而 data.a 只有 2 位，取 1000 的低 2 位，故 data.a 得值 0。

位段的定义和引用的说明：

(1)位段成员的类型必须指定为 unsigned 或 int 类型。

(2)若某一位段要从另一个字开始存放。可以用以下形式定义：

```
unsigned a: 1;
unsigned b: 2;     /* 一个存储单元 */
unsigned: 0;
unsigned c: 3;     /* 另一存储单元 */
```

长度为 0 的位段的作用是使下一个位段从下一个存储单元开始存放。

(3)一个位段必须存储在同一存储单元中，不能跨两个单元。如果第一个单元空间不能容纳下一个位段，则该空间不用，而从下一个单元起存放该位段。

(4)可以定义无名字段。如：

```
unsigned a: 1;
unsigned: 2;     /* 这两位空间不用 */
unsigned b: 3;
unsigned c: 4;
```

更多高清PDF图书 www.gqpdf.com

在 a 后面的是无名位段,该空间不用(图 9.9)。

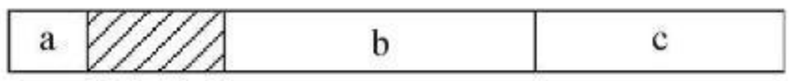


图 9.9　定义无名字段

(5)位段的长度不能大于存储单元的长度,也不能定义位段数组。

(6)位段可以用整型格式符输出,也可以用%u,%o,%x 等格式符输出。如:

```
printf("%d,%d,%d", data.a data.b data.c);
```

(7)位段可以在数值表达式中引用,会被系统自动地转换成整型数。

9.4　本章小结

高清PDF原创 www.gqpdf.com

位运算使得 C 语言能够直接运算到二进制位(bit),这也是 C 语言异于其他高级语言的地方,这使得 C 语言可能轻松地进行机器底层的编程。C 语言的位运算符主要有按位与(&)、按位或(|)、按位求反(~)、按位异或(^)、左移(<<)、右移(>>)六种。

课后练习

1. 编写一个函数,对一个 16 位的二进制数取出它的奇数位(即从左边起第 1、3、5、…、15 位)。

2. 编写一个函数用来实现左右循环移位。

第 10 章

文　件

高清PDF原创 www.gqpdf.com

【本章主要内容】

本章主要介绍 C 语言文件的定义、文件的打开与关闭、文件的读写等文件的基本操作。

【本章要求】

(1)了解 C 语言常用的文件操作函数。
(2)理解 C 语言文件的定义。
(3)掌握 C 语言文件的基本操作。

10.1　C 语言文件概述

10.1.1　C 文件的定义

活动 1　谈谈你对文件的认识。

请读者自己说出对文件的认识与了解,例如:什么是文件? 如何操作文件?

请读者上网利用搜索工具查询 C 语言对文件的定义并比较 C 文件定义跟读者自己的文件描述有何异同。

对计算机稍有了解的读者肯定会对文件有一定的认识,因为我们平时的计算机操作无时无刻不在用到文件,比如 Word 的使用,音乐的播放,程序的编写等等,所谓“文件”是指一组相关数据的有序集合。这个数据集有一个名称,叫做文件名。实际上在前面的各章中我们已经多次使用了文件,例如源程序文件、目标文件、可执行文件、库文件 (头文件)等。

文件通常是驻留在外部介质(如磁盘等)上的,在使用时才调入内存中来。从不同的角度可对文件作不同的分类。

更多高清PDF图书　www.gqpdf.com

(1)从用户的角度看,文件可分为普通文件和设备文件两种。

①普通文件是指驻留在磁盘或其他外部介质上的一个有序数据集,可以是源文件、目标文件、可执行程序;也可以是一组待输入处理的原始数据,或者是一组输出的结果。对于源文件、目标文件、可执行程序可以称作程序文件,对输入输出数据可称作数据文件。

②设备文件是指与主机相联的各种外部设备,如显示器、打印机、键盘等。在操作系统中,把外部设备也看作是一个文件来进行管理,把它们的输入、输出等同于对磁盘文件的读和写。

通常把显示器定义为标准输出文件,一般情况下在屏幕上显示有关信息就是向标准输出文件输出。如前面经常使用的 printf,putchar 函数就是这类输出。

键盘通常被指定标准的输入文件,从键盘上输入就意味着从标准输入文件上输入数据。scanf,getchar 函数就属于这类输入。

(2)从文件编码的方式来看,文件可分为 ASCII 码文件和二进制码文件两种。

①ASCII 文件也称为文本文件,这种文件在磁盘中存放时每个字符对应一个字节,用于存放对应的 ASCII 码。ASCII 码文件可在屏幕上按字符显示,例如源程序文件就是 ASCII 文件。由于是按字符显示,因此能读懂文件内容。

例如,数 5678 的存储形式为:

ASCII 码:　00110101　00110110　00110111　00111000

↓　↓　↓　↓

十进制码:　5　6　7　8

共占用 4 个字节。

②二进制文件是按二进制的编码方式来存放文件的。

例如,数 5678 的存储形式为:

00010110　00101110

只占二个字节。二进制文件虽然也可在屏幕上显示,但其内容无法读懂。C 系统在处理这些文件时,并不区分类型,都看成是字符流,按字节进行处理。

输入输出字符流的开始和结束只由程序控制而不受物理符号(如回车符)的控制。因此也把这种文件称作“流式文件”。

高清PDF原创 www.gqpdf.com

10.1.2　C语言的文件指针

在 C 语言中用一个指针变量指向一个文件,这个指针称为文件指针。通过文件指针就可对它所指的文件进行各种操作。

定义说明文件指针的一般形式为:

FILE * 指针变量标识符;

其中 FILE 应为大写,它实际上是由系统定义的一个结构,该结构中含有文件名、文件状态和文件当前位置等信息。在编写源程序时不必关心 FILE 结构的细节。

例如:

```
FILE * fp;
```

表示 fp 是指向 FILE 结构的指针变量,通过 fp 即可找到存放某个文件信息的结构变量,然后按结构变量提供的信息找到该文件,实施对文件的操作。习惯上也笼统地把 fp 称为指向一个文件的指针。

10.2 C 语言文件的基本操作

10.2.1 文件的打开



活动 2 请读者在文件定义的基础上思考文件有哪些基本的操作,这些基本的操作用程序如何实现?

熟悉 Windows 操作系统的读者应该对文件的基本操作比较了解:文件要使用之前必须打开;文件在使用之后必须关闭;要得到文件内容必须对文件进行读取;要修改文件内容必须对文件进行写入。即文件的基本操作包括了打开、关闭、读取和写入。

在 C 语言中,文件的基本操作都是由库函数来完成的。在本章内将介绍主要的文件操作函数。

所谓打开文件,实际上是建立文件的各种有关信息,并使文件指针指向该文件,以便进行其他操作。关闭文件则断开指针与文件之间的联系,也就禁止再对该文件进行操作。

fopen 函数用来打开一个文件,其调用的一般形式为:

文件指针名=fopen(文件名,使用文件方式);

其中,

"文件指针名"必须是被说明为 FILE 类型的指针变量。

"文件名"是被打开文件的文件名,为字符串常量或字符串数组。

"使用文件方式"是指文件的类型和操作要求。

例如:

```
FILE * fp;
fp=("filea","r");
```

其意义是在当前目录下打开文件 filea,只允许进行"读"操作,并使 fp 指向该文件。

又如:

```
FILE * fphzk
fphzk=("c:\\hzk16","rb")
```

其意义是打开 C 驱动器磁盘的根目录下的文件 hzk16,这是一个二进制文件,只允许按二进制方式进行读操作。两个反斜线"\\"中的第一个表示转义字符,第二个表示根目录。

使用文件的方式共有 12 种,表 10.1 给出了它们的符号和意义。

更多高清PDF图书 www.gqpdf.com

表 10.1　文件使用方式的表示方法

文件使用方式	意　义
“rt”	只读打开一个文本文件,只允许读数据
“wt”	只写打开或建立一个文本文件,只允许写数据
“at”	追加打开一个文本文件,并在文件末尾写数据
“rb”	只读打开一个二进制文件,只允许读数据
“wb”	只写打开或建立一个二进制文件,只允许写数据
“ab”	追加打开一个二进制文件,并在文件末尾写数据
“rt+”	读写打开一个文本文件,允许读写
“wt+”	读写打开或建立一个文本文件,允许读写
“at+”	读写打开一个文本文件,允许读,或在文件末追加数据
“rb+”	读写打开一个二进制文件,允许读写
“wb+”	读写打开或建立一个二进制文件,允许读写
“ab+”	读写打开一个二进制文件,允许读,或在文件末追加数据

高清PDF原创 www.gqpdf.com

对于文件使用方式有以下几点说明:

(1)文件使用方式由 r、w、a、t、b、+六个字符拼成,各字符的含义是:

r(read):　读
w(write):　写
a(append):　追加
t(text):　文本文件,可省略不写
b(banary):　二进制文件
+:　读和写

(2)凡用“r”打开一个文件时,该文件必须已经存在,且只能从该文件读出。

(3)用“w”打开的文件只能向该文件写入。若打开的文件不存在,则以指定的文件名建立该文件,若打开的文件已经存在,则将该文件删去,重建一个新文件。

(4)若要向一个已存在的文件追加新的信息,只能用“a”方式打开文件。但此时该文件必须是存在的,否则将会出错。

(5)在打开一个文件时,如果出错,fopen 将返回一个空指针值 NULL。在程序中可以用这一信息来判别是否完成打开文件的工作,并作相应的处理。因此常用以下程序段打开文件:

```
if((fp= fopen("c:\\hzk16","rb")==NULL)
{
  printf("\nerror on open c:\\hzk16 file!");
  getch();
  exit(1);
}
```

这段程序的意义是,如果返回的指针为空,表示不能打开 C 盘根目录下的 hzk16 文件,则给出提示信息“error on open c:\hzk16 file !”,下一行 getch()的功能是从键盘输入一个字符,但不在屏幕上显示。在这里,该行的作用是等待,只有当用户从键盘敲任一键时,程序才继续执行,因此用户可利用这个等待时间阅读出错提示。敲键后执行 exit(1)退出程序。

(6)把一个文本文件读入内存时,要将 ASCII 码转换成二进制码,而把文件以文本方式写入磁盘时,也要把二进制码转换成 ASCII 码,因此文本文件的读写要花费较多的转换时间。对二进制文件的读写不存在这种转换。

(7)标准输入文件(键盘),标准输出文件(显示器),标准出错输出(出错信息)是由系统打开的,可直接使用。

10.2.2 文件的关闭

文件一旦使用完毕,应用关闭文件函数把文件关闭,以避免文件的数据丢失等错误。C 语言使用 fclose 函数实现文件的关闭。fclose 函数调用的一般形式是:

fclose(文件指针);

例如:

```
fclose(fp);
```

正常完成关闭文件操作时,fclose 函数返回值为 0。如返回非零值则表示有错误发生。

10.2.3 文件的读取与写入

文件的读写有两种方式:顺序读写和随机读写。顺序读写即读写文件只能从头开始,顺序读写各个数据。但在实际问题中常要求只读写文件中某一指定的部分。为了解决这个问题可移动文件内部的位置指针到需要读写的位置,再进行读写,这种读写称为随机读写。

C 语言的文件读写函数都是由库函数来完成的,当然读者也可自定义函数来完成文件读写功能,文件读写常用的库函数我们将在 10.3 节进行介绍。

10.3 C 语言文件的常用读写函数

活动 3 文件的读取和写入是 C 语言文件的基本操作,作为 C 语言的库函数设计者,应该设计出方便 C 语言程序员使用的文件读取库函数。假设读者是 C 语言文件操作库函数的设计者,那么你需要设计哪一些类型的文件读写函数以方便程序员的使用?

我们知道,从文件编码的方式来看,文件可分为 ASCII 码文件和二进制码文件两种。在实际的应用中,如果文件是由 ASCII 码组成的,那么我们希望系统提供给我们可以读写一个字符的函数和可以读写整个字符串的函数。如果文件是由二进制码组成的,显然这时

更多高清PDF图书 www.gqpdf.com

以字符或字符串为单位来读写就没有意义了,因此,系统应该为我们提供整块数据的读写函数。有时,在实际应用中,我们又希望对具备一定格式的文件或内容进行读写,因此系统也必须提供格式化的读写函数,综上所述,C 语言为我们提供了如下几个常用的文件读写函数:

字符读写函数:fgetc 和 fputc。

字符串读写函数:fgets 和 fputs。

数据块读写函数:freed 和 fwrite。

格式化读写函数:fscanf 和 fprinf。

使用以上函数都要求包含头文件 stdio .h。下面我们对它们逐一进行介绍。

10 .3 .1　字符读写函数 fgetc 和 fputc

字符读写函数是以字符(字节)为单位的读写函数。每次可从文件读出或向文件写入一个字符。

1 .读字符函数 fgetc

fgetc 函数的功能是从指定的文件中读一个字符,函数调用的形式为:

字符变量=fgetc(文件指针);

例如:

```
ch= fgetc( fp);
```

其意义是从打开的文件 fp 中读取一个字符并送入 ch 中。

对于 fgetc 函数的使用有以下几点说明:

(1)在 fgetc 函数调用中,读取的文件必须是以读或读写方式打开的。

(2)读取字符的结果也可以不向字符变量赋值,

例如:

```
fgetc( fp);
```

但是读出的字符不能保存。

(3)在文件内部有一个位置指针。用来指向文件的当前读写字节。在文件打开时,该指针总是指向文件的第一个字节。使用 fgetc 函数后,该位置指针将向后移动一个字节。因此可连续多次使用 fgetc 函数,读取多个字符。应注意文件指针和文件内部的位置指针不是一回事。文件指针是指向整个文件的,须在程序中定义说明,只要不重新赋值,文件指针的值是不变的。文件内部的位置指针用以指示文件内部的当前读写位置,每读写一次,该指针均向后移动,它不需在程序中定义说明,而是由系统自动设置的。

【例 10 .1】　读入文件 c1 .doc,在屏幕上输出。

【程序】

```
voidmain()
{
  FILE * fp;
```

高清PDF原创 www.gqpdf.com

```
char ch;
if((fp= fopen("d:\\FILETEST\\1001.txt","rt"))==NULL)
{
    printf("\nCannot open file strike any key exit!");
    getch();
    exit(1);
}
ch= fgetc(fp);
while(ch!=EOF)
{
    putchar(ch);
    ch= fgetc(fp);
}
fclose(fp);
}
```

高清PDF原创 www.gqpdf.com

【分析】 本例程序的功能是从文件中逐个读取字符,在屏幕上显示。程序定义了文件指针 fp,以读文本文件方式打开文件“d:\\FILETEST\\1001.txt”,并使 fp 指向该文件。如打开文件出错,给出提示并退出程序。程序第 11 行先读出一个字符,然后进入循环,只要读出的字符不是文件结束标志(每个文件末有一结束标志 EOF)就把该字符显示在屏幕上,再读入下一字符。每读一次,文件内部的位置指针向后移动一个字符,文件结束时,该指针指向 EOF。执行本程序将显示整个文件。

2.写字符函数 fputc

fputc 函数的功能是把一个字符写入指定的文件中,函数调用的形式为:

fputc(字符量,文件指针);

其中,待写入的字符量可以是字符常量或变量,例如:

```
fputc('a',fp);
```

其意义是把字符 a 写入 fp 所指向的文件中。

对于 fputc 函数的使用也要说明几点:

(1)被写入的文件可以用写、读写、追加方式打开,用写或读写方式打开一个已存在的文件时将清除原有的文件内容,写入字符从文件首开始。如需保留原有文件内容,希望写入的字符以文件末开始存放,必须以追加方式打开文件。被写入的文件若不存在,则创建该文件。

(2)每写入一个字符,文件内部位置指针向后移动一个字节。

(3)fputc 函数有一个返回值,如写入成功则返回写入的字符,否则返回一个 EOF。可用此来判断写入是否成功。

【例 10.2】 从键盘输入一行字符,写入一个文件,再把该文件内容读出显示在屏幕上。

【程序】

```
voidmain()
```

更多高清PDF图书 www.gqpdf.com

```
{
  FILE * fp;
  char ch;
  if((fp= fopen("d:\\FILETEST\\1003.txt","wt+"))==NULL)
  {
      printf("Cannot open file strike any key exit!");
      getch();
      exit(1);
  }
  printf("input a string:\n");
  ch=getchar();
  while (ch!='\n')
  {
      fputc(ch,fp);
      ch=getchar();
  }
  rewind(fp);
  ch= fgetc(fp);
  while(ch!=EOF)
  {
      putchar(ch);
      ch= fgetc(fp);
  }
  printf("\n");
  fclose(fp);
}
```

【分析】 程序中第5行以读写文本文件方式打开文件1003.txt。程序第12行从键盘读入一个字符后进入循环,当读入字符不为回车符时,则把该字符写入文件之中,然后继续从键盘读入下一字符。每输入一个字符,文件内部位置指针向后移动一个字节。写入完毕,该指针已指向文件末。如要把文件从头读出,须把指针移向文件头,程序第18行rewind函数用于把fp所指文件的内部位置指针移到文件头。第19至24行用于读出文件中的一行内容。

10.3.2　字符串读写函数fgets和fputs

1.读字符串函数fgets

函数的功能是从指定的文件中读一个字符串到字符数组中,函数调用的形式为:

fgets(字符数组名,n,文件指针);

其中的n是一个正整数。表示从文件中读出的字符串不超过n－1个字符。在读入的最后一个字符后加上串结束标志'\0'。

高清PDF原创 www.gqpdf.com

例如：

```
fgets(str,n,fp);
```

的意义是从 fp 所指的文件中读出 n－1 个字符送入字符数组 str 中。

【例 10.3】 从 string 文件中读入一个含 10 个字符的字符串。

【程序】

```
voidmain()
{
  FILE * fp;
  char str[11];
  if((fp= fopen("d:\\FILETEST\\1002.txt","rt"))==NULL)
  {
      printf("\nCannot open file strike any key exit !");
      getch();
      exit(1);
  }
  fgets(str,11,fp);
  printf("\n%s\n",str);
  fclose(fp);
}
```

【分析】 本例定义了一个字符数组 str 共 11 个字节，在以读文本文件方式打开文件 string 后，从中读出 10 个字符送入 str 数组，在数组最后一个单元内将加上 '\0'，然后在屏幕上显示输出 str 数组。

对 fgets 函数有两点说明：

(1)在读出 n－1 个字符之前，如遇到了换行符或 EOF，则读出结束。

(2)fgets 函数也有返回值，其返回值是字符数组的首地址。

2.写字符串函数 fputs

fputs 函数的功能是向指定的文件写入一个字符串，其调用形式为：

fputs(字符串，文件指针)；

其中字符串可以是字符串常量，也可以是字符数组名，或指针变量，例如：

```
fputs("abcd",fp);
```

其意义是把字符串“abcd”写入 fp 所指的文件之中。

【例 10.4】 在例 10.2 中建立的文件 1003.txt 中追加一个字符串。

【程序】

```
voidmain()
{
  FILE * fp;
  char ch,st[20];
```

更多高清PDF图书 www.gqpdf.com

```
    if((fp=fopen("d:\\FILETEST\\1003.txt","at+"))==NULL)
    {
        printf("Cannot open file strike any key exit!");
        getch();
        exit(1);
    }
    printf("input a string:\n");
    scanf("%s",st);
    fputs(st,fp);
    rewind(fp);
    ch=fgetc(fp);
    while(ch!=EOF)
    {
        putchar(ch);
        ch=fgetc(fp);
    }
    printf("\n");
    fclose(fp);
}
```

10.3.3　数据块读写函数 fread 和 fwrite

C 语言还提供了用于整块数据的读写函数。可用来读写一组数据,如一个数组元素,一个结构变量的值等。

读数据块函数调用的一般形式为:

fread(buffer,size,count,fp);

写数据块函数调用的一般形式为:

fwrite(buffer,size,count,fp);

其中:

buffer 是一个指针,在 fread 函数中,它表示存放输入数据的首地址。在 fwrite 函数中,它表示存放输出数据的首地址。

size　表示数据块的字节数。

count　表示要读写的数据块块数。

fp　　表示文件指针。

例如:

```
fread(fa,4,5,fp);
```

其意义是从 fp 所指的文件中,每次读 4 个字节(一个实数)送入实数组 fa 中,连续读 5 次,即读 5 个实数到 fa 中。

【例 10.5】 从键盘输入两个学生数据,写入一个文件中,再读出这两个学生的数据显示在屏幕上。

【程序】

```
struct stu
{
   char name[10];
   int num;
   int age;
   char addr[15];
}boya[2],boyb[2],*pp,*qq;

void main()
{
  FILE *fp;
  char ch;
  int i;
  pp=boya;
  qq=boyb;
  if((fp=fopen("d:\\FILETEST\\1005","wb+"))==NULL)
  {
      printf("Cannot open file strike any key exit!");
      getch();
      exit(1);
  }
  printf("\ninput data\n");
  for(i=0;i<2;i++,pp++)
  scanf("%s%d%d%s",pp->name,&pp->num,&pp->age,pp->addr);
  pp=boya;
  fwrite(pp,sizeof(struct stu),2,fp);
  rewind(fp);
  fread(qq,sizeof(struct stu),2,fp);
  printf("\n\nname\tnumber    age    addr\n");
  for(i=0;i<2;i++,qq++)
  printf("%s\t%5d%7d    %s\n",qq->name,qq->num,qq->age,qq->addr);
  fclose(fp);
}
```

高清PDF原创 www.gqpdf.com

【分析】 本例程序定义了一个结构 stu,说明了两个结构数组 boya 和 boyb 以及两个结构指针变量 pp 和 qq。pp 指向 boya,qq 指向 boyb。程序以读写方式打开二进制文件“stu_list”,输入两个学生数据之后,写入该文件中,然后把文件内部位置指针移到文件首,读出两块学生数据后,在屏幕上显示。

即时练习

输入三个职工的信息(姓名,工号,职称,工资,所属部门)存放在文件中并在文件中读取

更多高清PDF图书 www.gqpdf.com

他们的信息最后打印到屏幕上。

高清PDF版制 www.gqpdf.com

10.3.4　格式化读写函数 fscanf 和 fprintf

fscanf 函数,fprintf 函数与前面使用的 scanf 和 printf 函数的功能相似,都是格式化读写函数。两者的区别在于 fscanf 函数和 fprintf 函数的读写对象不是键盘和显示器,而是磁盘文件。

这两个函数的调用格式为:

fscanf(文件指针,格式字符串,输入表列);

fprintf(文件指针,格式字符串,输出表列);

例如:

```
fscanf(fp,"%d%s",&i,s);
fprintf(fp,"%d%c",j,ch);
```

用 fscanf 和 fprintf 函数也可以完成例 10.5 的问题。修改后的程序如例 10.6 所示。

例 10.6　用 fscanf 和 fprintf 函数解决【例 10.5】的问题。

【程序】

```
struct stu
{
  char name[10];
  int num;
  int age;
  char addr[15];
}boya[2],boyb[2],*pp,*qq;

main()
{
  FILE *fp;
  char ch;
  int i;
  pp=boya;
  qq=boyb;
  if((fp=fopen("d:\\FILETEST\\1006","wb+"))==NULL)
  {
      printf("Cannot open file strike any key exit!");
      getch();
      exit(1);
  }
printf("\ninput data\n");
for(i=0;i<2;i++,pp++)
scanf("%s%d%d%s",pp->name,&pp->num,&pp->age,pp->addr);
```

```
    pp=boya;
    for(i=0;i<2;i++,pp++)
        fprintf(fp,"%s%d%d%s\n",pp->name,pp->num,pp->age,pp->addr);
    rewind(fp);
    for(i=0;i<2;i++,qq++)
        fscanf(fp,"%s%d%d%s\n",qq->name,&qq->num,&qq->age,qq->addr);
    printf("\n\nname\tnumber    age    addr\n");
    qq=boyb;
    for(i=0;i<2;i++,qq++)
        printf("%s\t%5d  %7d      %s\n",qq->name,qq->num, qq->age,qq->addr);
    fclose(fp);
}
```

【分析】 本程序中 fscanf 和 fprintf 函数每次只能读写一个结构数组元素,因此采用了循环语句来读写全部数组元素。还要注意指针变量 pp,qq 由于循环改变了它们的值,因此程序对它们重新赋予了数组的首地址。

10.3.5 文件的随机读写

1.文件的定位

实现随机读写的关键是要按要求移动位置指针,这称为文件的定位。

移动文件内部位置指针的函数主要有两个,即 rewind 函数和 fseek 函数。

rewind 函数前面已多次使用过,其调用形式为:

rewind(文件指针);

它的功能是把文件内部的位置指针移到文件首。

下面主要介绍 fseek 函数。

fseek 函数用来移动文件内部位置指针,其调用形式为:

fseek(文件指针,位移量,起始点);

其中:

"文件指针"指向被移动的文件。

"位移量"表示移动的字节数,要求位移量是 long 型数据,以便在文件长度大于 64KB 时不会出错。当用常量表示位移量时,要求加后缀"L"。

"起始点"表示从何处开始计算位移量,规定的起始点有三种:文件首,当前位置和文件尾。其表示方法如表 10.2 所示。

表 10.2 文件位置的表示方法

起始点	表示符号	数字表示
文件首	SEEK_SET	0
当前位置	SEEK_CUR	1
文件末尾	SEEK_END	2

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

高清PDF原创 www.gqpdf.com

例如：

```
fseek(fp,100L,0);
```

其意义是把位置指针移到离文件首100个字节处。

还要说明的是fseek函数一般用于二进制文件。在文本文件中由于要进行转换，故往往计算的位置会出现错误。

2.文件的随机读写

在移动位置指针之后，即可用前面介绍的任一种读写函数进行读写。由于一般是读写一个数据块，因此常用fread和fwrite函数。

【例10.7】 在文件1006中读出第二个学生的数据。

【程序】

```
struct stu
{
   char name[10];
   int num;
   int age;
   char addr[15];
}boy,* qq;

main()
{
   FILE * fp;
   char ch;
   int i=1;
   qq=&boy;
   if((fp= fopen("d:\\FILETEST\\1006","rb"))==NULL)
   {
       printf("Cannot open file strike any key exit!");
       getch();
       exit(1);
   }
   rewind(fp);
   fseek(fp,i* sizeof(struct stu),0);
   fread(qq,sizeof(struct stu),1,fp);
   printf("\n\nname\tnumber      age      addr\n");
   printf("%s\t%5d  %7d      %s\n",qq->name,qq->num,qq->age,qq->addr);
}
```

【分析】 文件1006已由例10.6的程序建立，本程序用随机读出的方法读出第二个学生的数据。程序中定义boy为stu类型变量，qq为指向boy的指针，以读二进制文件方式打开文件。其中的i值为1，表示从文件头开始，移动一个stu类型的长度，然后再读出的数据

即为第二个学生的数据。

即时练习

在 10.3.3 节即时练习所建立的职工文件中加入第 4 个职工的信息,要求第 4 个职工的信息在第 3 个职工之后,使用文件定位方法完成。

10.4 C 语言常用的文件检测函数

C 语言中常用的文件检测函数有以下几个。

1.文件结束检测函数 feof 函数

feof 函数调用格式:

feof(文件指针);

功能:判断文件是否处于文件结束位置,如文件结束,则返回值为 1,否则为 0。

2.读写文件出错检测函数

ferror 函数调用格式:

ferror(文件指针);

功能:检查文件在用各种输入输出函数进行读写时是否出错。如 ferror 返回值为 0 表示未出错,否则表示有错。

3.文件出错标志和文件结束标志置 0 函数

clearerr 函数调用格式:

clearerr(文件指针);

功能:本函数用于清除出错标志和文件结束标志,使它们为 0 值。

10.5 本章小结

C 系统把文件当做作一个“流”,按字节进行处理。文件按编码方式分为二进制文件和 ASCII 文件。C 语言中,用文件指针标识文件,当一个文件被打开时,可取得该文件指针。文件在读写之前必须打开,读写结束必须关闭。文件可按只读、只写、读写、追加四种操作方式打开,同时还必须指定文件的类型是二进制文件还是文本文件。文件可按字节,字符串,数据块为单位读写,文件也可按指定的格式进行读写。文件内部的位置指针可指示当前的读写位置,移动该指针可以对文件实现随机读写。

更多高清PDF图书 www.gqpdf.com

高清PDF原创 www.gqpdf.com

课后练习

1.C 语言中文件是如何定义的?

2.C 语言把文件分为哪些类型?

3.有两个磁盘文件 A 和 B,各存放一行字母,现要求把这两个文件中的信息合并(即文件 B 的内容连接到文件 A 后面)并追加到文件 C 中。

4.有 5 个学生,每个学生有 3 门课的成绩,从键盘输入以上数据(包括学号,姓名,三门课的成绩),计算出平均成绩,将原有数据和计算出的平均分数存放在磁盘文件“stud”中。

附 录 A

调试技巧

【本章主要内容】

本附录主要介绍 VC++6.0 程序调试的基本技巧和常用方法,并在此基础上介绍程序测试的基本方法。

【本章要求】

(1)理解程序测试的重要性。
(2)掌握程序调试的基本方法。

A.1 程序调试的基本方法

A.1.1 运行程序的方法

在 VC++6.0 编译环境里面运行 C 语言的程序可以遵循以下步骤:

1.打开 VC

开始→程序→Microsoft Visual Studio 6.0→Microsoft Visual C++6.0

2.新建

File(文件)菜单→New(新建)→Files(文件)→C++Source File(C++源文件),见图 X.1。

3.编译

Build(建立)菜单→Compile(编译,组合键:Ctrl+F7),在弹出的对话框中选择“是

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

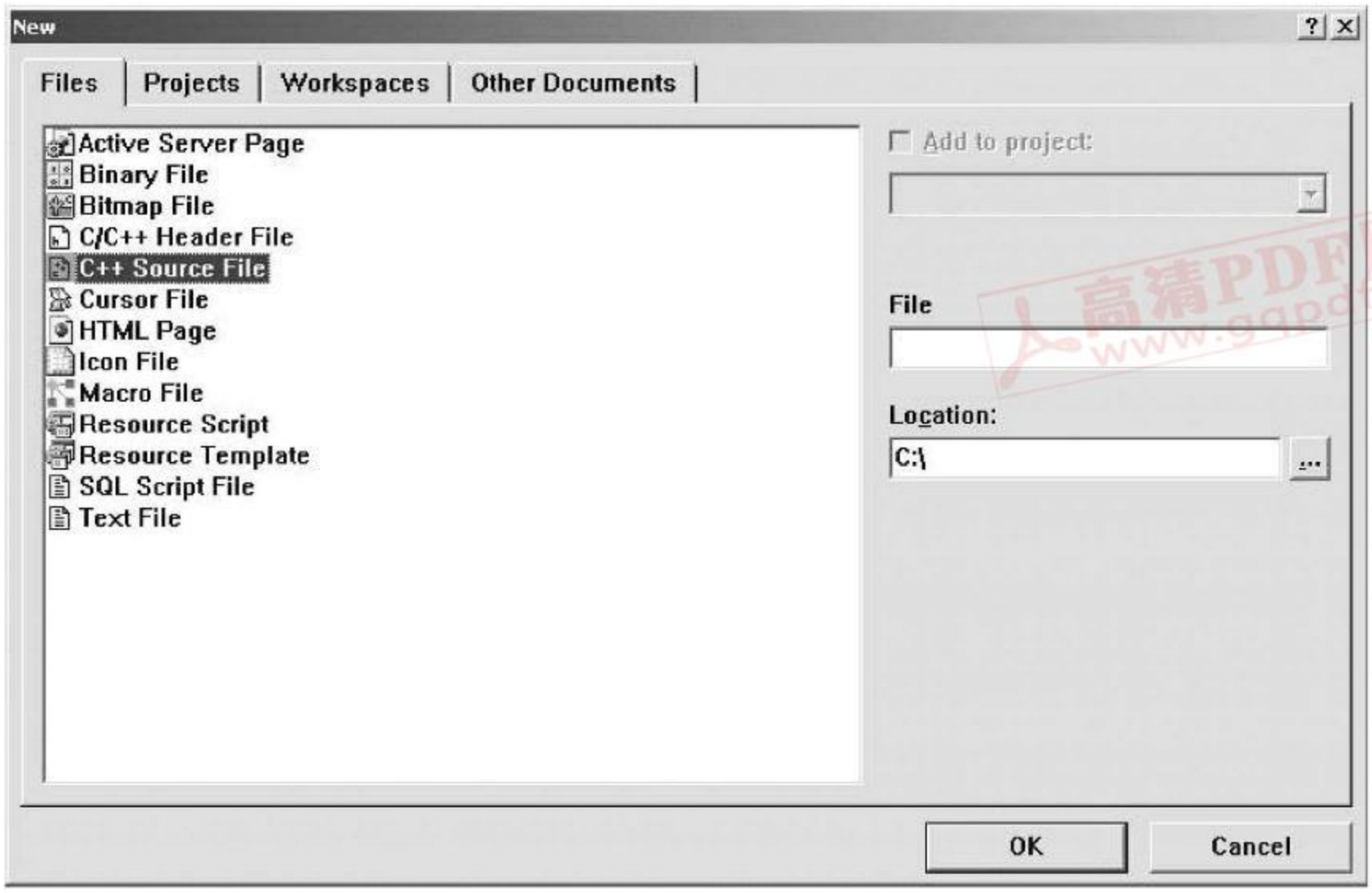


图 X.1　新建 C 源程序文件

(Y)”,即“create a default project workspace”(创建一个默认的项目工作空间),见图 X.2。



图 X.2　创建项目工作空间

在“保存为”对话框中选择源程序文件保存的位置和文件名(文件扩展名改为“.c”),见图 X.3。



图 X.3　保存源程序文件

如果源程序没有语法错误,“Output”(输出)框“Build”(建立)标签将显示“Cpp1 .obj—0 error(s), 0 warning(s)”(0 个错误,0 个警告)并生成目标(obj)文件,见图 X .4。

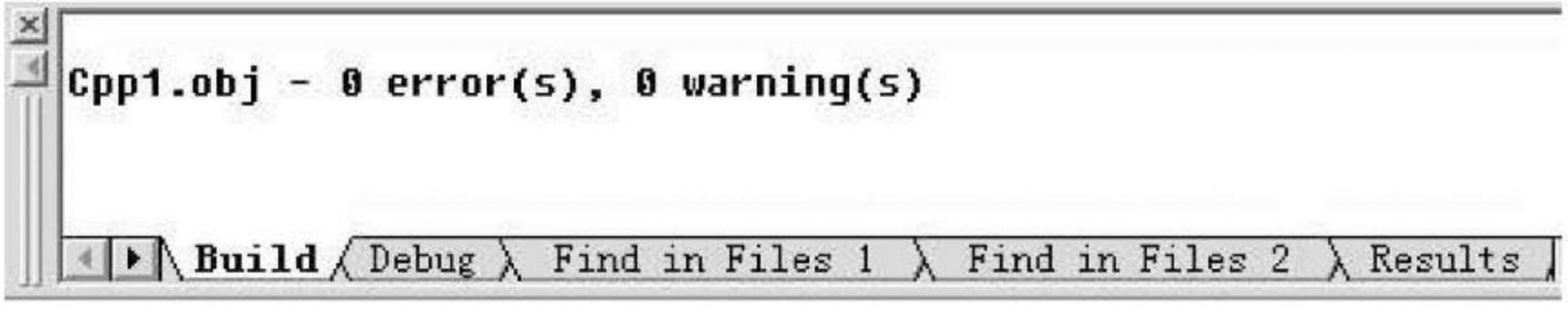


图 X .4 编译

4 .连接

Build 菜单→Build(组合键:F7),如果源程序没有语法错误,“Output”(输出)框“Build”(建立)标签将显示“Cpp1 .exe—0 error(s), 0 warning(s)”(0 个错误,0 个警告)并生成可执行(exe)文件,见图 X .5。



图 X .5 连接

5 .执行

Build 菜单→Execute(执行,组合键:Ctrl+F5),弹出一个控制台(命令行)窗口,显示程序的运行结果或者接收用户的输入,见图 X .6。

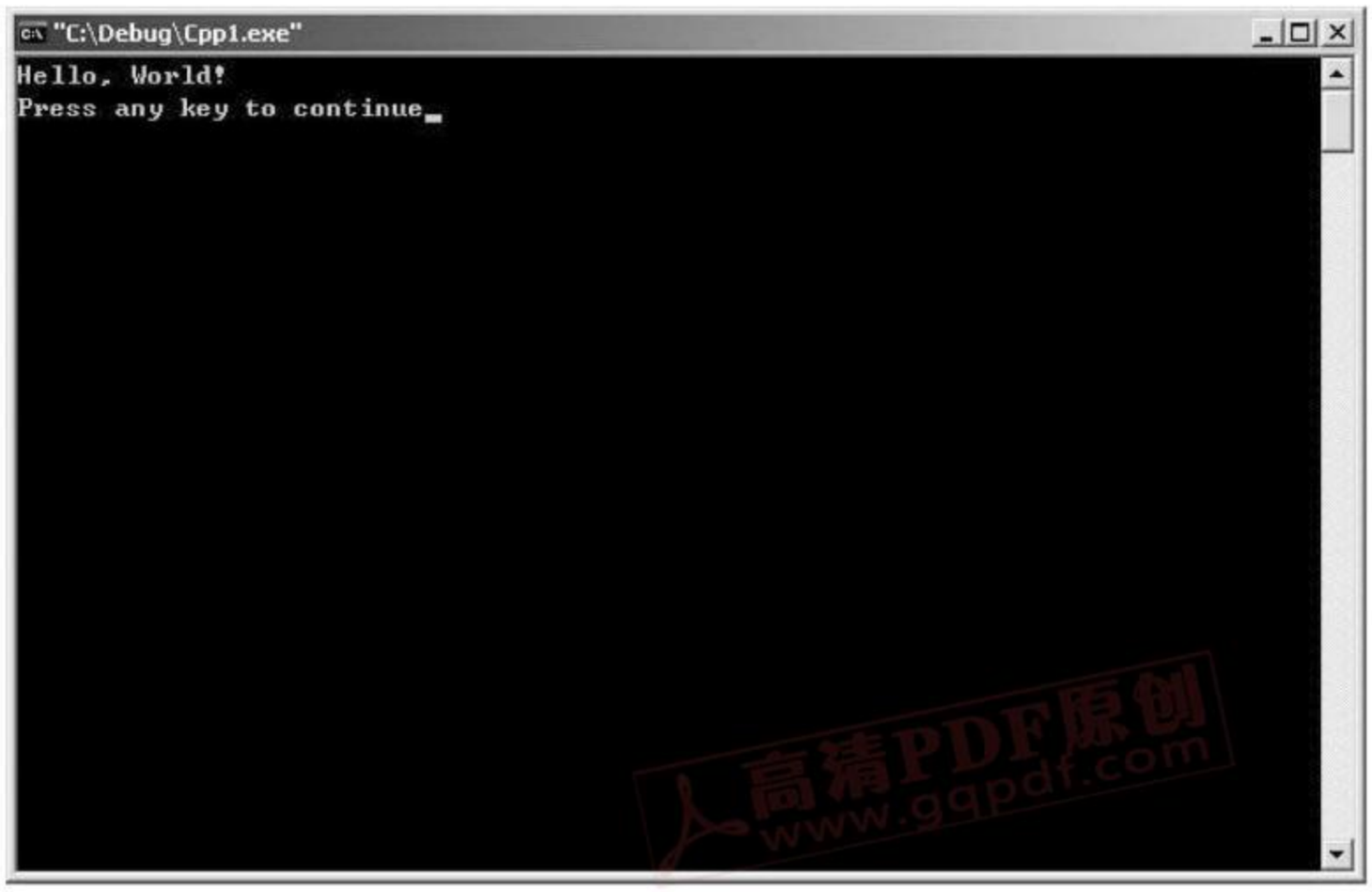


图 X .6 运行窗口

更多高清PDF图书 www.gqpdf.com

6.注意

当要编辑新的程序时,必须先关闭当前已经打开的 Workspace(File 菜单→Close Workspace(关闭工作空间)),然后才能新建文件。

A.1.2　调试程序的工具

1.调试菜单的使用

我们在 VC++6.0 环境中可以使用调试菜单上的各种功能来完成我们的程序调试。调试菜单只有在程序调试过程中才会动态的出现,调试菜单如图 X.7 所示:

除了调试菜单,VC 在程序调试过程中还会出现调试工具栏,方便我们更好地进行程序调试,调试工具栏如图 X.8 所示:

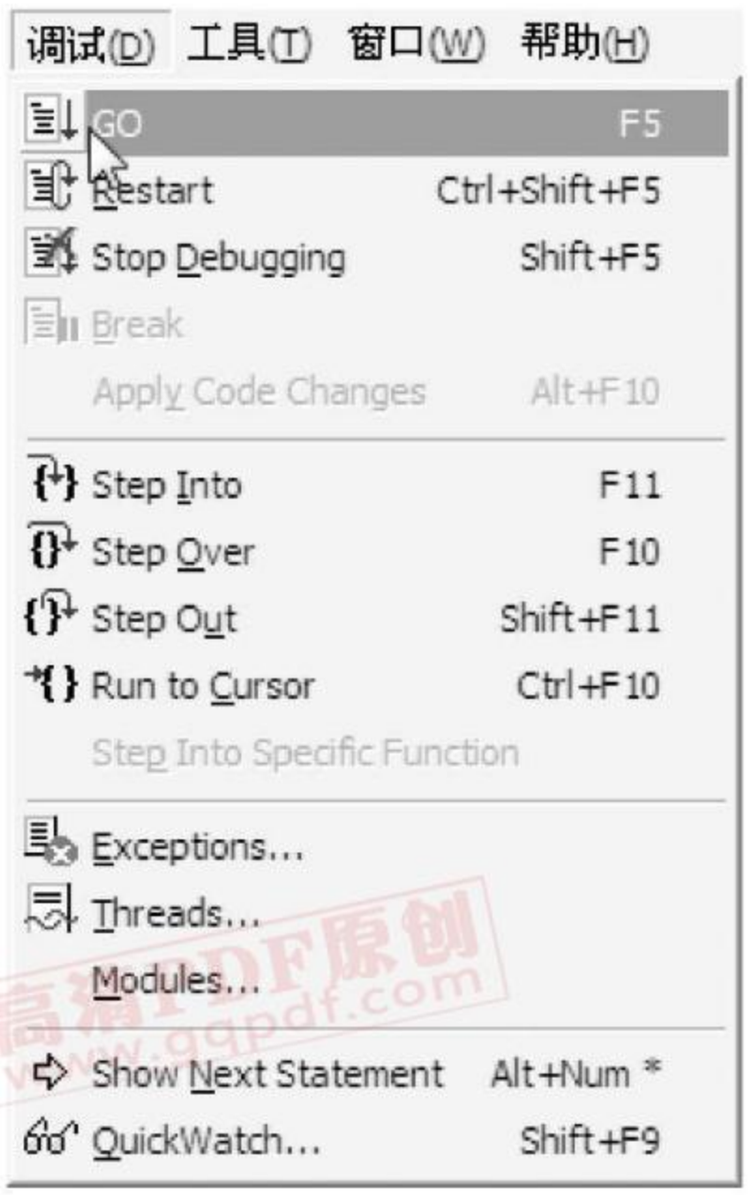


图 X.7　调试菜单



图 X.8　调试工具栏

下面我们介绍一下调试菜单的常用功能:

(1) Restart(Ctrl+shift+F5)

此功能将从程序的开始(第一有效行)处全速执行,而不是从当前所跟踪的位置开始调试,这时所有变量的当前值都将被丢弃,debugger 会自动停在程序的 main()开始处。这时如果选择 Step Over(F10)就可以逐步执行 main()函数了。

(2)  Stop Debugging(Shift+F5)

此功能将终止(所有)调试,并返回到常规编辑状态。

(3)  Break

此功能常常在遇到调用函数的语句时可见。它将在调试过程中的 debugger 当前位置挂起程序的执行,然后就可以在调试状态一修改程序的代码,接着可以用 Apply Code Changes(Alt+F10)来应用修改的代码到正在调试的程序当中。如果,当前需要等待(比如从 DOS 窗口输入值),则挂起后将不能再输入。

(4)  Apply Code Changes(Alt+F10)

此功能可以在程序正在调试程序过程中应用(挂起)修改后的源代码。如,选择 Break 功能并修改代码后,只要选择 Apply Code Changes(Alt+F10)就能将修改后的代码应用到正在调试的程序当中。

(5)  Show Next Statement(Alt+Num *)

此功能将显示程序代码的下一条语句,如果在源代码中找不到,则在 Disassembly 窗口中显示语句。当在  Disassembly 窗口中显示时,可以单击 Disassembly 返回到源代码窗口。

(6)  Step Into(F11)

此功能可以单步进入到在调试过程中所跟踪的调用函数的语句的函数内部。如,当前语句是"d=Display();",选择 Step Into(F11)后,Debugger 将进入 Display 函数内部并停在 Display 函数内部的第一条语句上,(此时,就可以选择 Step Over(F10)对 Display 函数进行单步调试了)。

(7)  Step Over(F10)

此功能可以单步对所在函数单步调试,如果调试的语句是一个调用函数的语句时,Debugger将全速执行所调用的函数,单步(一步)通过所调用的函数,Debugger 停在该调用语句的下一条语句上。

(8)  Step Out(Shift+F11)

此功能将使 Debugger 切换回全速执行到被调用函数结束,并停在该函数调用语句的下一条语句上。当确定所调用的函数没有问题时可以用这个功能全速执行被调用函数。

(9)  Run to Cursor(Ctrl+F10)

此功能将全速执行到包含插入点光标所在的行,可以作为在插入点光标处设置常规断点的一种选择。注意,当光标处不是一个有效的执行语句时此功能将不起作用。

(10)  Go(F5)

此功能将全速执行程序直到遇到一个断点或程序结束,或直到程序暂停等待用户输入。注意,此功能可有效的调试循环,常将断点设置在循环体内,重复的按 F5 全速执行循环体

更多高清PDF图书 www.gqpdf.com

可以测试循环过程中的产生的变化。

(11)Step Into Specific Function

此功能可以单步通过程序中的指令,并进入指定的函数调用,此功能对于函数的嵌套层不限。

2.运行与调试程序常用的快捷键

程序调试菜单功能众多,初学的读者可能一时难以掌握。其实我们在调试程序的时候只是使用其中一部分常用的功能,而且在使用这些功能的时候经常用快捷键来完成,读者只要记住表 X.1 的常用快捷键并使用它们进行程序调试的训练即可掌握程序调试的基本方法。

表 X.1　调试常用快捷键

单步进入(进入函数内部执行)	F11
单步执行	F10
单步跳出(退出函数内部)	SHIFT+F11
运行到光标	CTRL+F10
设置断点	F9
清除断点	CTRL+SHIFT+F9
运行到断点	F5
Compile(编译,生成 .obj 文件)	CTRL+F7
Build(组建,先 Compile 生成 .obj 再 Link 生成 .exe 文件)	F7
编译并运行	CTRL+F5
停止调试	SHIFT+F5

A.1.3　程序调试举例

程序调试是指对程序的查错和排错。一般来说程序的错误分为语法错误和逻辑错误两种,如果是语法错误,编译器在程序运行之前便会通知程序员。语法错误多是由于程序员的疏忽或者程序基本功不扎实造成的,这里我们不作讨论。另一类是逻辑错误,或称算法错误。这类错误的隐蔽性较高,当问题的规模比较庞大或者算法比较复杂时便容易出现此类错误,我们程序调试的目的从根本上来说主要是要解决逻辑错误。而逻辑错误的发现与解决需要技巧和长期程序调试积累起来的经验,下面我们以例 X.1 来说明。

【例 X.1】　冒泡排序法的程序调试

```
void Bubblesort()
{
   int a[10];
   int i, j, t;
```

高清PDF原创 www.gqpdf.com

```
printf("input 10 numbers:\n");
for(i=0; i< 10; i++)
    scanf("%d",&a[i]);

printf("\n");

for(j=0; j< 9; j++)
    for(i=0; i< j; i++)
        if(a[i]> a[i+1])//大数沉底
        {
            t=a[i];
            a[i]=a[i+1];
            a[i+1]=t;
        }

printf("the sorted numbers:\n");

for(i=0; i< 10; i++)
    printf("%d",a[i]);

printf("\n");
}
voidmain()
{
  Bubblesort();
}
```

我们已经在数组里面讲过冒泡排序的算法,它的基本原理是将相邻两个数比较,将小的调到前头(小数上浮,大数下沉)。我们将程序输入之后按 CTRL+F5 编译并运行,然后输入无序的 10 个数可以看到运行的结果如图 X.9 所示。

显然这不是我们期待的结果,程序可以运行说明语法没错,运行结果不对说明逻辑上出了问题。我们发现做排序的关键是 Bubblesort 函数,因为主函数里面只有一条语句便是调用这个函数。我们可以按尝试以下几种调试方法:

(1)按 F10 执行单步运行,在执行单步运行的时候我们发现程序仅仅把 Bubblesort 函数当成一条语句执行,显然这种方法并不是我们需要的。

(2)按 F11 进入函数内部执行,我们发现在程序运行到 Bubblesort 函数时会自动进入函数内部运行,如图 X.10 所示:

继续按 F11 的时候它会逐条语句执行,这时候它运行到了 printf 函数,printf 是系统函数,这时如果继续按 F11 就会发现程序会自动进入 printf 函数,如图 X.11 所示:

我们可以看到系统函数内部的执行情况,不过这对我们的调试并没有帮助,可以按 SHIFT+F11 跳出函数继续执行下一条语句。但是一般来说我们会选择函数中关键的语句开始执行而不是从第一行便开始运行,所以第二种方法(进入函数单步执行)必须配合第三

更多高清PDF图书 www.gqpdf.com

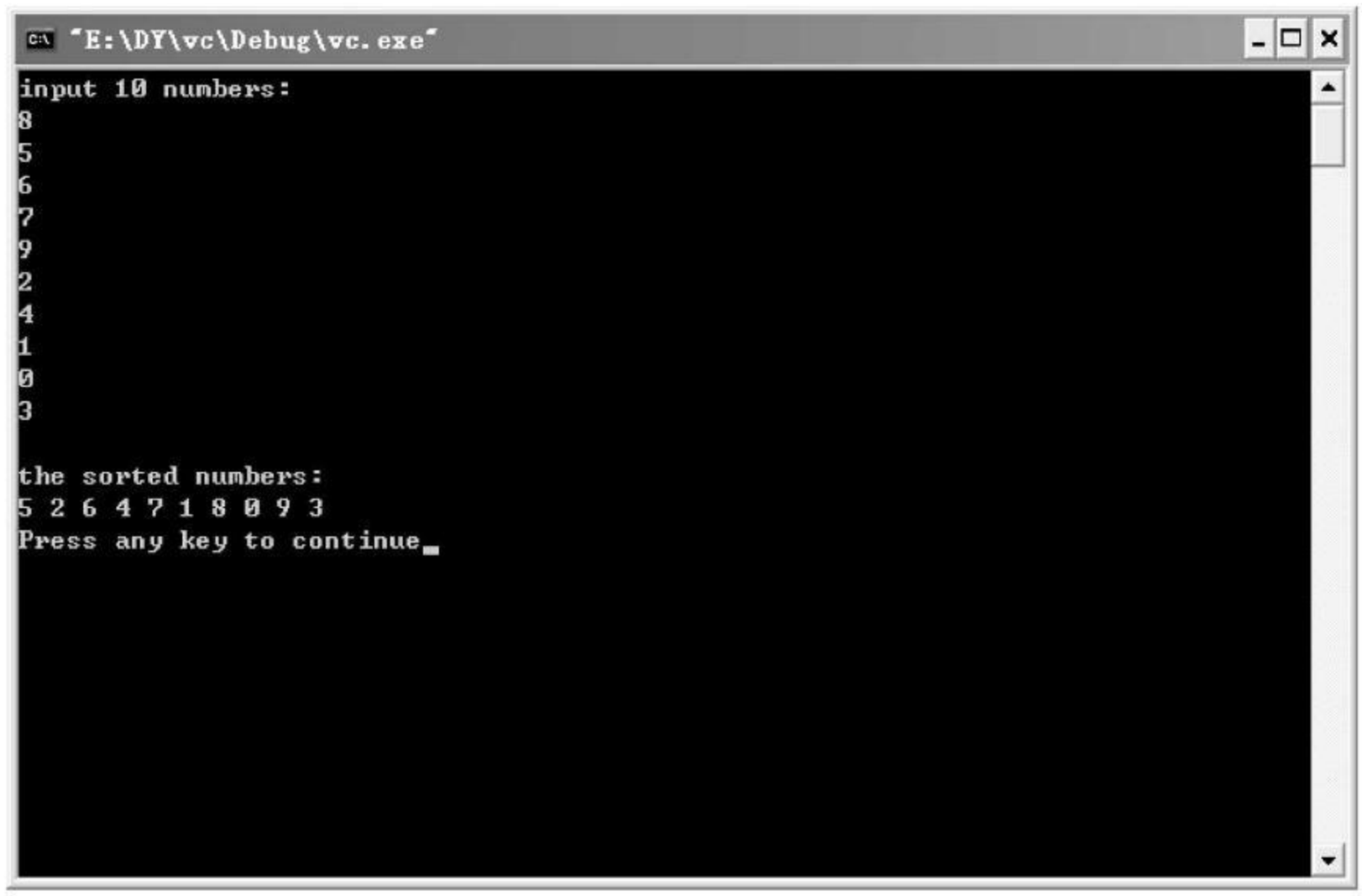


图 X.9　错误的冒泡程序运行结果

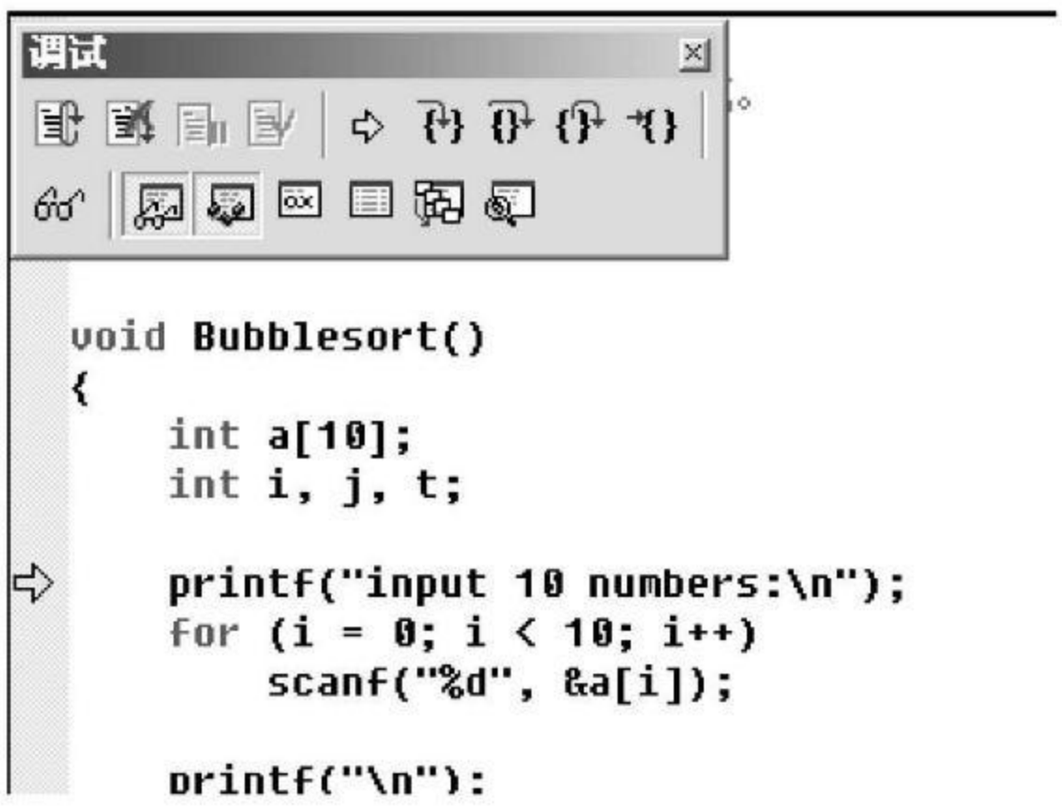


图 X.10　按 F11 进入函数内部执行

种方法(设置断点)才能达到效果。

(3)按 F9 设置断点可以根据我们的需要选择程序开始执行的位置,在【例 X.1】中我们发现前面的输入并不需要我们调试,后面的输出也不用。我们所要考虑的只是中间的排序算法的问题,所以我们可以在第一个 for 那里设置断点,如图 X.12 所示:

然后按 F5 运行到该断点,程序提示让我们输入 10 个数字,输入 10 个数字之后程序便出现黄色的执行箭头,如图 X.13 所示:

继续按 F10 我们可以发现黄色箭头往下移,我们在调试工具栏里面找到 watch 按钮,发现它是按下的状态,然后观察下面的部分,出现了两个窗体如图 X.14 所示:

左边的窗体是自动给出的动态跟踪窗体,右边的窗体便是 watch 窗体(监视窗体),我们

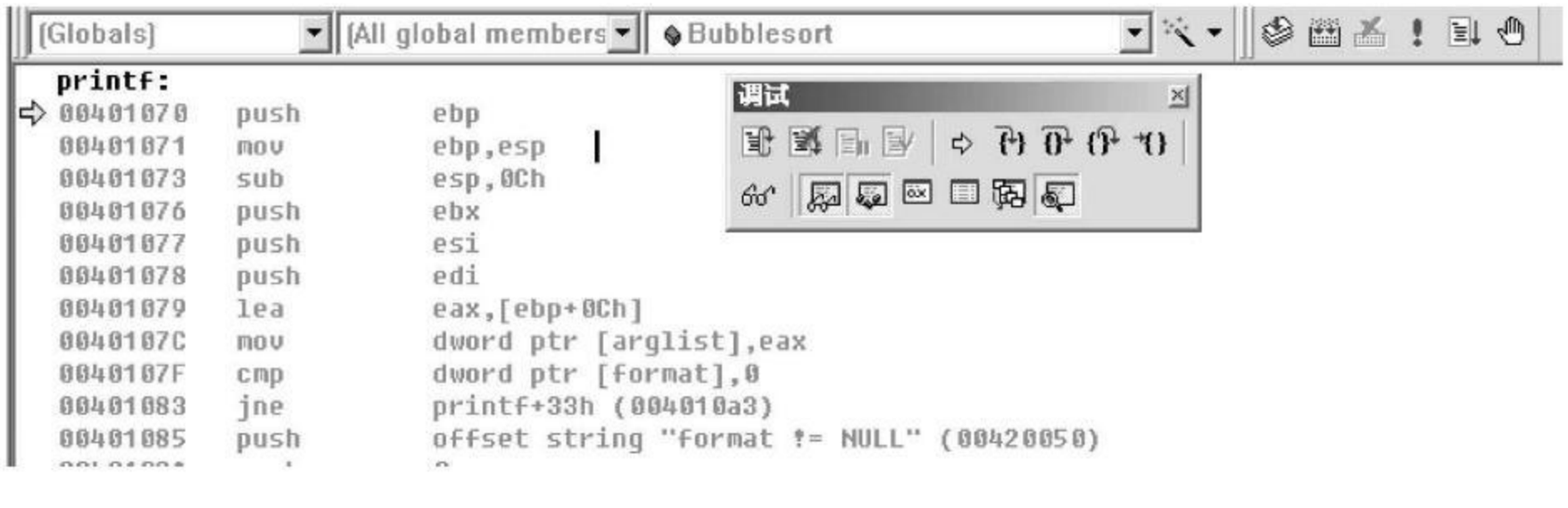


图 X.11　进入 printf 函数继续单步运行

```
printf("input 10 numbers:\n");
for (i = 0; i < 10; i++)
    scanf("%d", &a[i]);

printf("\n");

for(j = 0; j < 9; j++)
    for(i = 0; i < j; i++)
        if (a[i] > a[i+1])//大数沉底
        {
            t = a[i];
            a[i] = a[i + 1];
            a[i + 1] = t;
        }
```

图 X.12　在第一个 for 循环那里设置断点

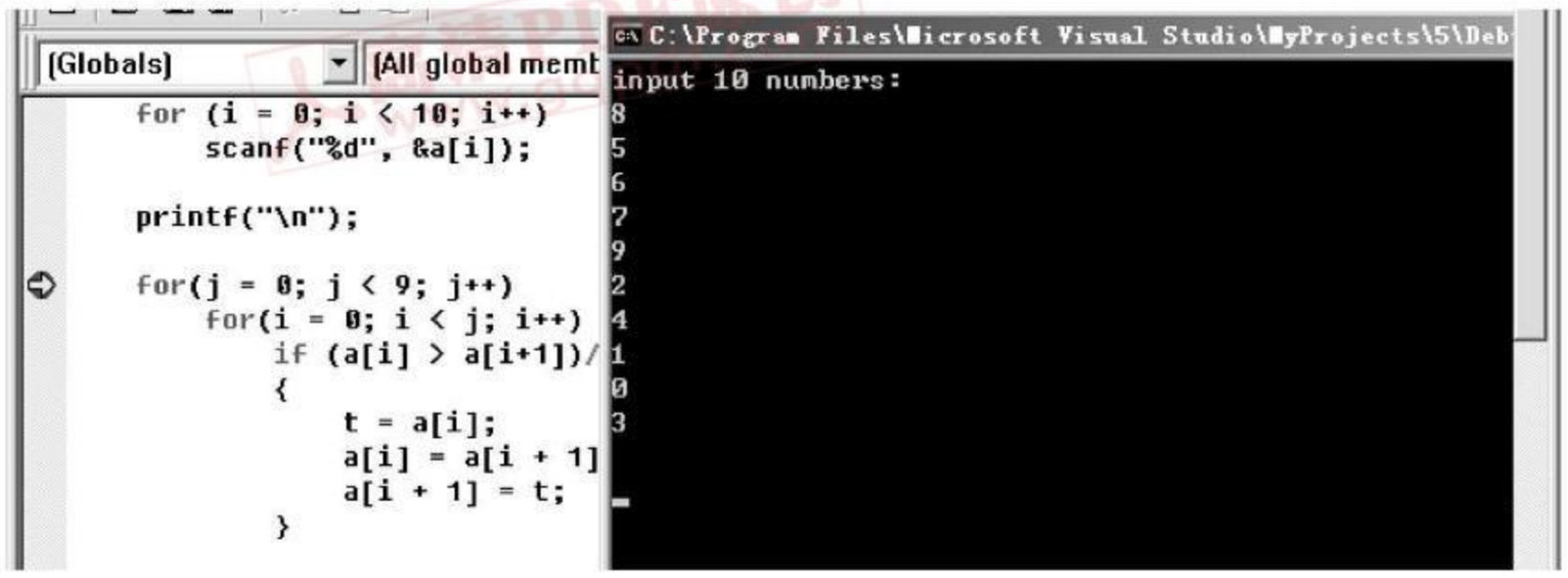


图 X.13　输入数字后开始单步执行

可以发现有四个 watch 窗体即从 watch1 到 watch4，在监视窗体中我们可以在 name 的那一列输入变量的名称。如图 X.14 所示我们输入了 i 和 j，它们初始值分别为 10 跟 0，那么以后程序继续执行时这两个变量就会自动改变它们的值方便我们观察。

在算法中我们发现变量 t 和数组元素 a[i]、a[i+1]也是需要观察的，所以我们把它们也添加到监视窗体中，接下来便开始单步执行，我们这里列出前面的部分变量中间值，见表 X.2：

更多高清PDF图书 www.gqpdf.com



图 X.14　watch 窗口

表 X.2　监视窗口中变量的中间值

i	j	t	a[i]	a[i+1]	备　注
0	0	—858993460	8	5	初始值
0	1	—858993460	8	5	执行 if 语句之前
0	1	8	5	8	执行 if 语句之后
1	1	8	8	6	执行 if 语句之前
1	2	8	8	6	不执行 if 语句
0	2	8	5	8	执行 if 语句之前
0	2	8	5	8	不执行 if 语句
1	2	8	8	6	执行 if 语句之前
1	2	8	6	8	执行 if 语句之后
2	2	8	8	7	执行 if 语句之前
2	2	8	8	7	不执行 if 语句
0	3	8	5	6	执行 if 语句之前
0	3	8	5	6	不执行 if 语句
1	3	8	6	8	执行 if 语句之前
1	3	8	6	8	不执行 if 语句
2	3	8	8	7	执行 if 语句之前
2	3	8	7	8	执行 if 语句之后
3	3	8	8	9	执行 if 语句之前
3	3	8	8	9	完成第 4 次外循环

高清PDF原创 www.gqpdf.com

表 X.2 给出了 4 次外循环(即 j=0 到 3 时的循环)的中间变量的值。其他外循环读者可依照此做法得出中间变量的值。在研究中间变量的值时我们发现程序总是将满足 i<j 条件的 a[i]和 a[i+1]作比较,然后将大者调到前面,外循环是从 0 到 9,所以当 j 值不断增大时我们发现每一次都是从 0 开始,这也就造成了重复的比较,显然在逻辑上出现了错误。正确的想法是每比较一趟外循环(假设有 n 个数),最大的数就沉底,留下 n-1 个数继续比较,直到全部比较完毕。所以经过分析,我们得到正确的程序书写方法应该是把第二个循环(即内循环)的 for 语句改为:for(i=0; i<10-j; i++),这样便符合大数沉底的要求。

A.2 程序测试的基本方法

A.2.1 程序测试的必要性

程序测试是软件测试的一个组成部分,我们知道一个软件是有若干个程序模块有机组成的一个系统,系统的运行是否正常取决于每个程序模块的健壮性。而每个程序模块又是由若干程序段组合而成,所以我们在写程序的过程中要注意使程序具备健壮、可维护性高、可移植性高等特点。一个很小的纰漏可能会造成整个软件系统的崩溃和无法挽救的损失,所以程序的测试相当重要,它是软件开发过程的一个重要环节,是程序员必备的一项基本素质。

A.2.2 程序测试举例

程序测试方法众多,目前常用的有黑盒法和白盒法,所谓黑盒法即忽略程序内部的代码进行整体的测试,而白盒法则是将程序内部结构进行分析并完成程序所有分支的测试。我们学习 C 语言的程序调试就是使用白盒法,一般情况下我们会两种方法结合以达到较好的效果,下面举一使用黑盒法测试程序的例子。

【例 X.2】 某班进行期末考试,按考试成绩分为优秀(90 分及以上)、良好(80 分及以上,90 分以下)、中等(70 分及以上,80 分以下)、及格(60 分及以上,70 分以下)和不及格(60 分以下)五个等级。要求编写程序输入某个同学的成绩并输出该同学的成绩等级。

该题目的程序设计对于具备一定 C 语言基础的读者并不困难。关键是如何测试我们的程序,这里我们使用黑盒测试所以并不需要考虑代码,我们的主要任务是找出符合测试要求的数据。测试的时候最好是将输入、输出与结论列成一张表格。

很多 C 语言的初学者在进行此程序测试的时候便是考虑每个等级输入一个分数,然后看输出结果正确与否,如表 X.3 所示。

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

表 X.3　测试数据 1

输　入	输　出	结　论
95	优秀	正确
87	良好	正确
72	中等	正确
63	及格	正确
45	不及格	正确

在做了表 X.3 的测试之后很多人可能会下结论:此程序运行正确。其实,我们忽略了一个重要的问题,就是边界问题,读者可以设想一下有没有同学刚好考 80 分呢? 如果是这种情况那么是属于中等还是良好呢? 根据题目的要求应该是显示良好的,因为良好包括 80 分。所以在测试的时候我们除了一般条件,还应该考虑边界条件,这就是边界测试。

如表 X.4 所示,我们将所有边界数据加进去之后发现输出的结果都为正确的,那么是不是可以就此下结论说我们的程序是正确的呢? 其实,在实际应用中输入是不能够绝对预知的,输入的东西是不是一定是数字呢? 可能有人输入错误把某个字母当成数字输入进去了,这时我们应该如何处理呢? 所以测试时的数据应该考虑越出边界的情况(包括输入非法字符的情况)。

表 X.4　测试数据 2(包含边界测试)

输　入	输　出	结　论
100	优秀	正确
95	优秀	正确
90	优秀	正确
87	良好	正确
80	良好	正确
72	中等	正确
70	中等	正确
63	及格	正确
60	及格	正确
45	不及格	正确
0	不及格	正确

如表 X.5 所示,我们将非法输入作为测试数据加了进来,如果结果正确(比如输出提示非法输入),则说明程序具有一定的健壮性。但是在实际应用中还应考虑到数据类型的问题,比如输入 59.999999,那么系统又是如何判断的呢? 在 C 语言里面如果用浮点数据类型,那么整数是被某个近似的浮点数来判断的,比如 60 被 59.999999 代替,一般情况下在单科考试中不会出现此情况,但是在算平均分的时候就务必要注意了。而且,测试数据并不是

高清PDF原创 www.gqpdf.com

每一种只用一个就可以说明问题,有时必须要有一定数量的数据才能反映问题,所以我们在做程序设计的时候需要考虑得更全面,才能使我们的程序更健壮,更不容易出错。

表 X.5 测试数据 3(包含非法输入)

输 入	输 出	结 论
100	优秀	正确
95	优秀	正确
90	优秀	正确
87	良好	正确
80	良好	正确
72	中等	正确
70	中等	正确
63	及格	正确
60	及格	正确
45	不及格	正确
0	不及格	正确
a	?	?
T31	?	?
152	?	?
—13	?	?

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

附 录 B

常用 ASCII 对照表

ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符
0	NUT	32	(space)	64	@	96	、
1	SOH	33	!	65	A	97	a
2	STX	34	”	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	,	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	—	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DCI	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	X	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	\|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	—	127	DEL

高清PDF原创 www.qqpdf.com

附 录 C

C 语言常用库函数

应包含库文件 math .h

函数类型	函数形式	功　能	返回类型
数学函数	abs(int i)	求整数的绝对值	int
	fabs(double x)	返回浮点数的绝对值	double
	floor(double x)	向下舍入	double
	fmod(double x , double y)	计算 x 对 y 的模 , 即 x/y 的余数	double
	exp(double x)	指数函数	double
	log(double x)	对数函数 ln(x)	double
	log10(double x)	对数函数 log	double
	labs(long n)	取长整型绝对值	long
	modf(double value , double * iptr)	把数分为指数和尾数	double
	pow(double x , double y)	指数函数(x 的 y 次方)	double
	sqrt(double x)	计算平方根	double
	sin(double x)	正弦函数	double
	asin(double x)	反正弦函数	double
	sinh(double x)	双曲正弦函数	double
	cos(double x)	余弦函数	double
	acos(double x)	反余弦函数	double
	cosh(double x)	双曲余弦函数	double
	tan(double x)	正切函数	double
	atan(double x)	反正切函数	double
	tanh(double x)	双曲正切函数	double

高清PDF原创 www.gqpdf.com

更多高清PDF图书 www.gqpdf.com

续表

应包含头文件 string.h

函数类型	函数形式	功　能	返回类型
字符串函数	strcat(char * dest,const char * src)	将字符串 src 添加到 dest 末尾	char
	strchr(const char * s,int c)	检索并返回字符 c 在字符串 s 中第一次出现的位置	char
	strcmp(const char * s1,const char * s2)	比较字符串 s1 与 s2 的大小,并返回 s1—s2	int
	stpcpy(char * dest,const char * src)	将字符串 src 复制到 dest	char
	strdup(const char * s)	将字符串 s 复制到最近建立的单元	char
	strlen(const char * s)	返回字符串 s 的长度	int
	strlwr(char * s)	将字符串 s 中的大写字母全部转换成小写字母,并返回转换后的字符串	char
	strrev(char * s)	将字符串 s 中的字符全部颠倒顺序重新排列,并返回排列后的字符串	char
	strset(char * s,int ch)	将一个字符串 s 中的所有字符置于一个给定的字符 ch	char
	strspn(const char * s1,const char * s2)	扫描字符串 s1,并返回在 s1 和 s2 中均有的字符个数	char
	strstr(const char * s1,const char * s2)	描字符串 s2,并返回第一次出现 s1 的位置	char
	strtok(char * s1,const char * s2)	检索字符串 s1,该字符串 s1 是由字符串 s2 中定义的定界符所分隔	char
	strupr(char * s)	将字符串 s 中的小写字母全部转换成大写字母,并返回转换后的字符串	char

应包含并头文件 ctype.h

函数类型	函数形式	功　能	返回类型
字符函数	isalpha(int ch)	若 ch 是字母('A'—'Z','a'—'z')返回非 0 值,否则返回 0	int
	isalnum(int ch)	若 ch 是字母('A'—'Z','a'—'z')或数字('0'—'9')返回非 0 值,否则返回 0	int
	isascii(int ch)	若 ch 是字符(ASCII 码中的 0—127)返回非 0 值,否则返回 0	int
	iscntrl(int ch)	若 ch 是作废字符(0x7F)或普通控制字符(0x00—0x1F)返回非 0 值,否则返回 0	int
	isdigit(int ch)	若 ch 是数字('0'—'9')返回非 0 值,否则返回 0	int
	isgraph(int ch)	若 ch 是可打印字符(不含空格)(0x21—0x7E)返回非 0 值,否则返回 0	int
	islower(int ch)	若 ch 是小写字母('a'—'z')返回非 0 值,否则返回 0	int
	isprint(int ch)	若 ch 是可打印字符(含空格)(0x20—0x7E)返回非 0 值,否则返回 0	int
	ispunct(int ch)	若 ch 是标点字符(0x00—0x1F)返回非 0 值,否则返回 0	int
	isspace(int ch)	若 ch 是空格(' '),水平制表符('\t'),回车符('\r'),走纸换行('\f'),垂直制表符('\v'),换行符('\n'),返回非 0 值,否则返回 0	int

高清PDF原创 www.gqpdf.com

续表

应包含并头文件 ctype .h

函数类型	函数形式	功　能	返回类型
字符函数	isupper(int ch)	若 ch 是大写字母('A'—'Z')返回非 0 值,否则返回 0	int
	isxdigit(int ch)	若 ch 是 16 进制数('0'—'9','A'—'F','a'—'f')返回非 0 值,否则返回 0	int
	tolower(int ch)	若 ch 是大写字母('A'—'Z')返回相应的小写字母('a'—'z')	int
	toupper(int ch)	若 ch 是小写字母('a'—'z')返回相应的大写字母('A'—'Z')	int

应包含头文件 stdio .h

函数类型	函数形式	功　能	返回类型
输入输出函数	getch()	从控制台(键盘)读一个字符,不显示在屏幕上	int
	putch()	向控制台(键盘)写一个字符	int
	getchar()	从控制台(键盘)读一个字符,显示在屏幕上	int
	putchar()	向控制台(键盘)写一个字符	int
	getchar()	从控制台(键盘)读一个字符,显示在屏幕上	int
	getc(FILE * stream)	从流 stream 中读一个字符,并返回这个字符	int
	putc(int ch,FILE * stream)	向流 stream 写入一个字符 ch	int
	getw(FILE * stream)	从流 stream 读入一个整数,错误返回 EOF	int
	putw(int w,FILE * stream)	向流 stream 写入一个整数	int
	fclose(handle)	关闭 handle 所表示的文件处理	FILE *
	fgetc(FILE * stream)	从流 stream 处读一个字符,并返回这个字符	int
	fputc(int ch,FILE * stream)	将字符 ch 写入流 stream 中	int
	fgets(char * string,int n,FILE * stream)	流 stream 中读 n 个字符存入 string 中	char *
	fopen(char * filename,char * type)	打开一个文件 filename,打开方式为 type,并返回这个文件指针,type 可为以下字符串加上后缀	FILE *
	fputs(char * string,FILE * stream)	将字符串 string 写入流 stream 中	int
	fread(void * ptr,int size,int nitems,FILE * stream)	从流 stream 中读入 nitems 个长度为 size 的字符串存入 ptr 中	int
	fwrite(void * ptr,int size,int nitems,FILE * stream)	向流 stream 中写入 nitems 个长度为 size 的字符串,字符串在 ptr 中	int
	fscanf(FILE * stream,char * format[,argument,…])	以格式化形式从流 stream 中读入一个字符串	int
	fprintf(FILE * stream,char * format[,argument,…])	以格式化形式将一个字符串写给指定的流 stream	int
	scanf(char * format[,argument,…])	从控制台读入一个字符串,分别对各个参数进行赋值,使用 BIOS 进行输出	int
	printf(char * format[,argument,…])	发送格式化字符串输出给控制台(显示器),使用 BIOS 进行输出	int

更多高清PDF图书 www.gqpdf.com

参考文献

[1]谭浩强著.C程序设计(第三版).北京:清华大学出版社,2005
[2]何钦铭主编.C语言程序设计.北京:人民邮电出版社,2003
[3]张引等主编.C程序设计基础课程设计.浙江:浙江大学出版社,2007

高清PDF原创 www.gqpdf.com

高清PDF原创 www.gqpdf.com

教师反馈表

感谢您一直以来对浙大版图书的支持和爱护。为了今后为您提供更好、更优秀的计算机图书，请您认真填写下面的意见反馈表，以便我们对本书做进一步的改进。如果您在阅读过程中遇到什么问题，或者有什么建议，请告诉我们，我们会真诚为您服务。如果您有出书需求，以及好的选题，也欢迎来电来函。

填表日期：____年____月____日

教师姓名		所在学校名称			院　系	
性　　别	□男　□女	出生年月		职　务	职　称	
联系地址				邮　编	办公电话	
				手　机	家庭电话	
E-mail				QQ/MSN		

您是通过什么渠道知道本书的

□书店　□经人推荐　□网站介绍　□图书目录　□其他_____

您从哪里购买本书的

□书店　□网站　□邮购　□学校统一订购□其他_____

您对本书的总体感觉是

□很满意　□满意　□一般　□不满意　原因_____

具体来说，您觉得本书的封面设计　□很好　□还行　□不好　□很差_____

您觉得本书的纸张及印刷　□很好　□还行　□不好　□很差_____

您觉得本书的技术含量　□很高　□还可以　□一般　□很低　□极低_____

您觉得本书的内容设置　□很好　□还可以　□一般　□不太好　□很差_____

您觉得本书的实用价值　□很高　□还可以　□一般　□很低　□极低_____

目前主要教学专业、科研领域方向

	主授课程	教材及所属出版社	学生人数	教材满意度
课程一：				□满意　□一般　□不满意
课程二：				□满意　□一般　□不满意

教学层次：	□中职中专　□高职高专　□本科　□硕士　□博士 其他：_____
希望我们与您经常保持联系的方式（划√）	□电子邮件信息　□定期邮寄书目　□定期电话咨询 □定期登门拜访　□通过教材科联络　□通过编辑联络

教材出版信息

方向一		□准备写　□写作中　□已成稿　□已出版　□有讲义
方向二		□准备写　□写作中　□已成稿　□已出版　□有讲义

填表说明：本表可以直接邮寄至：杭州市天目山路148号浙江大学西溪校区内浙江大学出版社理工事业部

联系人：胡　伟　电话：0571—88925595　手机：13454177008　传真：0571—88925590

吴昌雷　电话：0571—88273342　手机：13675830904　E-mail：changlei_wu@zju.edu.cn

更多高清PDF图书 www.gqpdf.com